ANO II - N.º 19
ABRIL 1983
Cr$ 500,00
ISSN 0101-3041

Micro Sistemas

A PRIMEIRA REVISTA BRASILEIRA DE MICROCOMPUTADORES

Ano II
Nº 19
Abril 1983

Micro Sistemas

SUMÁRIO

SEÇÕES

editorial

• No editorial do número passado, falávamos na necessidade de efetuarmos, enquanto nação, um esforço no sentido de maiores investimentos para a área de recursos humanos.

Isto porque entendemos ser esta ação fundamental para não só comprarmos, mas igualmente adaptarmos e desenvolvermos de forma criativa e produtiva a tecnologia.

• Na esfera do usuário particular, também um esforço para aprofundar os conhecimentos é necessário, embora utilizar um microcomputador não seja tarefa difícil.

Utilizar o micro não é difícil. Também não é suficiente. Utilizar bem é o "x" da questão. Ler o primeiro capítulo do manual e comprar algumas fitas de jogos já é o bastante para que o usuário se divirta. Porém é uma subutilização imperdoável de um equipamento poderoso, que dá a seu usuário possibilidades interminá-veis de se aprofundar e partir para aplicações de todo tipo.

• MICRO SISTEMAS tem procurado dar a seus leitores subsídios para que se desenvolvam. E neste número, fugindo um pouco à nossa composição geral, que inclui matérias variadas, fizemos uma edição de consulta, bastante completa, sobre um tema amplo: as linguagens de programação.

No futuro certamente não precisaremos de tantas regras. Poderemos programar em nossa linguagem natural, dizendo diretamente à máquina o que esperamos dela, uma vez que esta já terá perfeito conhecimento e intimidade com a complexa linguagem humana.

Mas enquanto o homem e o computador não dividem uma linguagem comum; enquanto este só compreende suas sequências de zeros e uns, existem muitas linguagens nas quais podemos programá-los. Umas mais voltadas a aplicações comerciais, outras ideais para cálculos científicos. Algumas mais utilizadas em computadores pessoais do que outras. Umas já consideradas ultrapassadas e outras que ainda estão sendo estudadas.

Como a maioria dos usuários de micros em nosso país utiliza o BASIC, esta linguagem é constantemente abordada em MS. Agora, mostramos diversas outras, que poderão vir a interessar nossos leitores, em função das aplicações específicas que eles tenham em mente.

Procuramos apresentar as principais linguagens através de pequenos artigos que não têm outras pretensões senão introduzi-las aos leitores. Aquelas que não foram apresentadas separadamente constam do artigo "As linguagens de programação". Quanto ao BASIC, neste número você encontrará um artigo que compara os comandos desta linguagem em equipamentos compatíveis com Apple, TRS-80 e ZX, da Sinclair. Estamos esperando as cartas com a opinião dos leitores sobre esta edição. Boa leitura.

Alda Campos

Editor/Diretor Responsável:
Alda Surerus Campos

REDAÇÃO:
Beatriz Carolina Gonçalves, Denise Pragana, Edna Araripe, Maria da Glória Esperança, Nelson Guimarães, Paulo Henrique de Noronha, Ricardo Inojosa, Stela Lachtermacher.

Assessoria Técnica:
Luiz Antônio Pereira, Newton Duarte Braga Jr., Orson Voerckel Galvão, Paulo Saldanha.

Colaboradores: Amaury Moraes Jr., Arnaldo Milstein Mefano, Cláudio Curotto, Edson Espírito Santo, Fausto Arinos de Almeida Barbuto, Ivo D'Aquino Neto, Jôneson Carneiro de Azevedo, Liane Tarouco, Luciano Nilo de Andrade, Renato Degiovani, Renato Sabbatini.

Supervisão Gráfica:
Lázaro Santos
Diagramação:
Silvio Sola
Arte-final:
Jorge Nacari
Fotografia: Carlão Limeira, Monica Leme, Nelson Jurno.
Ilustração: Hubert, Jorge Nacari, Willy.

ADMINISTRAÇÃO: Claudia Lara Campos, Lais Denise Menezes, Wilma Ferreira Cavalcanti, Elizabeth Lopes dos Santos, Tânia Cévolo Gonçalves.

PUBLICIDADE
São Paulo:
Natal Calina
Al. Gabriel Monteiro da Silva, 1229
CEP 01441 - Tel.: (011) 280-4144
Rio de Janeiro:
Marcus Vinícius da Cunha Valverde
Av. Almirante Barroso, 90 - grupo 1114
CEP 20031 - Tel.: (021) 240-8297

CIRCULAÇÃO E ASSINATURAS:
Francisco Rufino Siqueira (RJ), Marcos dos Passos Neves (RJ), Dilma Menezes da Silva (RJ), Maria Izilda Guastaferro (SP), Janio Aparecido S. Pereira (SP).

DISTRIBUIÇÃO:
A. S. Motta - Imp. Ltda.
Tels.: (021) 252-1226 e 263-1560 - RJ;
(011) 288-5932 - SP.

Composição:
Gazeta Mercantil S.A.

Fotolito:
Beni Ltda.

Impressão e Acabamento:
Cia. Lithographica Ypiranga

Tiragem:
45 mil exemplares

Assinaturas:
No país: 1 ano - Cr$ 5.000,00

MICRO SISTEMAS é uma publicação mensal da

ATU Análise, Teleprocessamento e Informática Editora Ltda.

Diretor Presidente:
Álvaro Teixeira Assumpção
Diretor Vice-Presidente:
Alda Surerus Campos
Diretor:
Roberto Rocha Souza Sobrinho
Endereços:
Al. Gabriel Monteiro da Silva, 1229
Jardim Paulistano - São Paulo - SP - CEP 01441
Tel.: (011) 280-4144

Av. Almirante Barroso, 90 - grupos 1103 e 1114
Centro - Rio de Janeiro - RJ - CEP 20031
Tel.: (021) 240-8297

cartas

O sorteado deste mês, que receberá gratuitamente uma assinatura de um ano de MICRO SISTEMAS, é Carlos Roberto Cavalcante, da Paraíba.

RESPOSTA

Atendemos às solicitações dos amigos Augusto de Souza, de Salvador, e José Wesley Costa Matias, de Fortaleza (ambos publicados em MICRO SISTEMAS nº 15, na Seção Cartas sob o título "SUGESTÕES", em que pediam maior número de jogos para micros em MS), estou remetendo um microprograma, que mostro aos meus amigos que me pedem para demonstrar o meu sistema: o Sinclair ZX81 com 16 Kb.
Andrew Fairbairn
São Paulo-SP

É isso mesmo Andrew, gostamos de ver a sua intenção de colaborar com a gente. O seu jogo "Viagem aos Cosmos" está sendo examinado por nossa assessoria e, assim que tivermos uma resposta, entramos em contato com você. Esperamos que outros leitores sigam o seu excelente exemplo.

MS AGRADECE

Adquiri recentemente um microcomputador DGT-100, mas já há algum tempo venho lendo os exemplares desta revista, fato que, inclusive, me ajudou na decisão desta compra.

Quero parabenizá-los pelo trabalho sério e consciente desenvolvido por esta equipe.
Waldemar J. Folli
Vila Velha- ES

Quero parabenizá-los pela garra com que lançaram a MS no mercado, e por vir mantendo esta ótima qualidade que já nos é conhecida.

Ivan C. de Castilho
Itajaí-SC

Cumpre-nos vir à presença de V. Sas. agradecer a circulação da matéria "As fábricas de jogos", notadamente pelo interesse exclusivamente informativo e imparcial de V. Sas., beneficiando leitores e fabricantes.

Nossos sinceros agradecimentos e elevados votos de sucesso à sua publicação.
Micron Eletrônica Com. Ind. Ltda.
São José dos Campos-SP

Em reconhecimento a grande rapidez no envio das revistas que estavam em atraso, tenho o prazer de agradecer a simpática equipe de MICRO SISTEMAS para que isto também sirva de incentivo na continuidade de real atenção para com os seus assinantes.
Etelvino C. de S. Júnior
Salvador-BA

O motivo pelo qual escrevo esta carta é para cumprimentá-los pela explicação sobre a linguagem de máquina no DGT-100, editado no número 15 desta revista.

Paulo Henrique Cançado
Curitiba-PR

JOYSTICK PARA TODOS

"JOYSTICK", com este título publicamos na Seção de Cartas de MICRO SISTEMAS 16 (edição de janeiro de 83), uma carta do leitor Carlos Eduardo T. de Menezes, do Espírito Santo, na qual solicitava informações sobre a possibilidade de adquirir um par de joystick para o CP-500 da Prológica. Recebemos em nossa redação uma amável carta do leitor Enrique Ferri que, entre outras sugestões, pede-nos para informar ao Carlos Eduardo como adaptar o joystick ao seu equipamento, conforme reproduzimos:

Com respeito à carta do colega Carlos Menezes, de Cachoeiro de Itapemirim, Espírito Santo, sugiro procurar o artigo da revista BYTE, de março de 1977, onde o artigo "An Inexpensive Joystick Interface" dá todos os detalhes para ligar um conjunto de joystick a qualquer microcomputador.

Sendo sua revista pioneira no ramo, gostaria, dentro de suas diretrizes, que publicassem um artigo sobre o sistema UNIX; a linguagem Ada; e ainda sobre os sistemas de desenvolvimento nos quais são realizados os principais micros comerciais e industriais de uso profissional.

Enrique H. H. Ferri
São Paulo-SP

MANUAL DO ZX81

Na Seção de Cartas de MICRO SISTEMAS 17 (edição de fevereiro de 83), publicamos sob o título "ZX81", carta do leitor Amós Moreira de Oliveira, de São Paulo, em que nos pedia informações sobre como obter um manual do ZX81 em Português. Gentilmente, o leitor Carlos Leal, de Santa Catarina, nos escreveu a seguinte carta:

Gostaria de informar ao Amós, e à revista também, que existe o manual do ZX81 em Português. Ele foi traduzido e editado, acompanhando opcionalmente o micro, pelo importador da Sinclair em Lisboa tendo, inclusive, a mesma capa que o original Inglês.

No entanto, o manual em Inglês é muito bem feito e extremamente simples e didático, de forma que um mínimo conhecimento da língua e um bom dicionário resolvem perfeitamente o problema.

O nome e endereço do importador em Lisboa é: Laudry Ltda., Rua Tomás da Anunciação 53-A, 1300, Lisboa, Portugal. Mas cuidado! Não peça para eles lhe mandarem esse manual porque, com certeza, o farão esperar um longo tempo. Infelizmente não são de confiança. Tente conseguir alguém que lhe traga diretamente.
Carlos Leal
Florianópolis-SC

PARA A SPLICE VIA MS

Desejo fazer público meu reconhecimento à firma Splice, fabricante do xadrez eletrônico brasileiro "BYTE". No dia 12 de fevereiro comprei o jogo e, mesmo faltando um peão branco decidi levá-lo, pois era sábado à noite e o xadrez era o último existente na loja.

No dia 18, após usar durante alguns dias um botão em substituição ao peão, enviei uma carta ao fabricante solicitando informação sobre como eu poderia adquirir a peça faltante. Perguntei também se o aquecimento verificado no canto do tabuleiro era normal.

Como resposta, no dia 23 recebi uma gentil carta do gerente de engenharia da Splice, eng. Paulo Roberto Freitas de Carvalho, anexando o peão branco gratuitamente, explicando que o aquecimento do suporte é normal e colocando-se ao meu dispor para quaisquer esclarecimentos.

Isto, além de uma finíssima atenção, revela um alto grau de eficiência e organização, e um nobre respeito pelo consumidor.

Portanto, solicito a gentileza de publicar esta carta em sua conceituada revista MICRO SISTEMAS.
Juan Jorge Thierer
São Paulo-SP

SUGESTÕES

Uma revista especializada como a MICRO SISTEMAS era o que estava faltando para atingir pessoas de centros menos desenvolvidos em Informática, como o em que eu resido. Daí meu entusiasmo ao me deparar com MS, apressando-me em transmitir aos senhores minhas impressões.

Que incluam nas páginas da revista alguns programas para lazer, como jogos e passatempos, é a minha sugestão.

Erinalva Lins C. da Silva
Natal-RN

Mando uma sugestão: por que vocês não colocam cada mês um estudo ou algumas informações sobre as linguagens existentes no mundo da computação? Gostaria de escrever mais, mas não teria linhas para escrever tudo o que esta revista merece, pois uma revista de alta competência deve ser sempre elogiada e falada. Como todos dizem, é uma revista Made in Brazil.

Mário Gomes da Silva
São Paulo-SP

Não é do meu feitio escrever cartas para revistas, mas MICRO SISTEMAS tem feito por onde merecer um grande elogio, tanto quanto à qualidade do material como também os assuntos veiculados pela mesma. Continuem assim, pois é disso que estamos precisando, sangue novo para tocar tudo neste país, que a cada ano tem se atrasado em relação ao resto do mundo. Potencial nós temos, e é o que se vê com sua bem estruturada revista.

Gostaria ainda de sugerir que o caderno de micros disponíveis no mercado se tornasse um encarte especial tal qual já conhecemos existir em outras revistas (não do ramo), o que ajudaria bastante ao brasileiro na decisão para a escolha de um micro, pessoal ou não. Poder-se-ia até incluir periféricos em tal encarte.

Pedro Paulo Luz C. Filho
Brasília-DF

Tenho lido quase todos os números de MICRO SISTEMAS e apesar de ter gostado da maioria dos artigos, senti falta de uma maior ênfase nos assuntos que tratam do funcionamento interno de um microcomputador. Acharia interessante que publicassem matérias sobre tecnologias de fabricação de circuitos impressos ou sobre o interior de microprocessadores mais conhecidos, como a que foi feita com o 6502, escrita por Carlos E. Tarrisse da Fontoura, que achei muito interessante.

No mais, só tenho a parabenizá-los por essa revista que se dinamiza a cada mês.
Eduardo D. de Oliveira
Rio Claro-SP

Quero parabenizar a equipe de MICRO SISTEMAS pelos ótimos artigos publicados.

Acho que muitos iniciantes, como eu, gostariam que nos programas publicados por MS fossem citadas as capacidades mínimas de memória RAM exigidas pelos mesmos. Também gostaria que programas da área de entretenimento para o TK82-C saíssem nesta, que é a melhor revista sobre Informática.
Marcelo S. Ferraz
Campinas-SP

Gostaria de sugerir que vocês escrevessem mais artigos sobre o TK82-C, tais como jogos, jogos educacionais, sistemas comerciais como Conta Bancária, Controle de Estoque etc., já que tenho um TK82-C.

Gostaria também que republicassem o Curso de BASIC, pois ganhei o computador há pouco e só tenho a revista do mês de janeiro de 83.
Marcelo Ris
São Paulo-SP

Inicialmente quero parabenizar toda a equipe pela edição de janeiro. Edições com jogos, comparações entre micro nacionais, dicas etc. são assuntos que agradam a leitores de qualquer atividade profissional.

Gostaria, portanto, de sugerir:

1. Criar uma seção mensal para jogos e micro mercado;
2. Criar uma edição anual ou semestral sobre jogos, dando prioridade aos micros nacionais mais vendidos;
3. Fazer comparação direta entre dois micros (nacionais ou não) concorrentes no mercado, fornecendo as particularidades de cada um.

Eloísa Pacheco Lima
São Paulo-SP

Que tal criarem um artigo especializado em "macetes de programação" para os diversos micros brasileiros? Um artigo sobre programação, com informações que os manuais dos equipamentos não dão, ou quando comentam, o fazem de modo incompleto. Um artigo aberto àquelas "manias" que cada equipamento possui com relação à sua linguagem. Surgiu-me então uma idéia para o nome deste artigo: MICRO MACETES.

Tenho certeza de que será de grande ajuda aos usuários e programadores de micros, pois certos "macetes" levam muito tempo para serem descobertos. Esta era a sugestão. Espero que daqui por diante ela venha integrar sempre as páginas de MICRO SISTEMAS. Aproveito para parabenizá-los pela brilhante iniciativa do Curso de Assembler, e deixar as minhas "Micro Saudações" a todos os que lêem e fazem MICRO SISTEMAS.

Beno Vicente Schirmer
Hamburgo Velho-RS

Gostaria de cumprimentá-los pela edição desta tão conceituada revista, e certificá-los do grande interesse que a mesma desperta em nossa comunidade estudantil (alunos do curso de Tecnólogo em Processamento de Dados da UFPB).

Quero fazer duas sugestões para que, na medida do possível, sejam analisadas e enquadradas nos próximos números da revista: gostaria que vocês fizessem um artigo detalhado sobre técnicas e maneiras mais eficientes de funcionamento, ou até como se processa o armazenamento de informações medidas em unidades de bytes por polegadas, trilhas etc., e também os vários tipos e métodos de acesso a essas informações. Isso com relação a discos, fitas, tambor magnético e disquete.

A outra sugestão é que se viabilize um curso de linguagem PL/1 adaptada para micros.
Carlos Roberto Cavalcante
João Pessoa-PB

Envie suas sugestões para MICRO SISTEMAS. Elas serão anotadas em nossa pauta e procuraremos, na medida do possível, viabilizá-las.

Xadrez

Enxadrista experiente, Luciano Nilo de Andrade já escreveu para os jornais "Correio da Manhã" e "Data News", e para a revista "Fatos & Fotos". Luciano é economista, trabalha no Ministério da Fazenda e atualmente, escreve uma coluna no jornal carioca "Ultima Hora" todas as quintas-feiras. As opiniões e comentários de Luciano Nilo de Andrade estão sempre presentes em MICRO SISTEMAS toda vez que o assunto for Microcomputadores e Xadrez.

CHALLENGER 9, uma excelente compra

A comercialização do **Sensory Chess Challenger 9** por 140 dólares, em Nova Iorque, parece ser no momento e dentre os aparelhos que eu conheço a melhor compra. Isto, para quem não quer gastar centenas de dólares com aparelhos mais sofisticados que nem sempre apresentam melhor performance enxadrística.

A qualidade do jogo do **Challenger 9** é muito boa e pode ser observada no exemplo que apresentamos nesta coluna. E ainda, quando o confrontamos com outro aparelho comercializado por mais do dobro de seu preço, ele não teve dificuldades em derrotá-lo.

Os comandos do **Challenger 9** têm a simplicidade característica dos produtos da **Fidelity.** Seu rating (superior a 1700) é um dado honesto, sem exageros publicitários. Para os adictos, existe como adicional dois cartuchos (cartridges) de aberturas a um preço superior ao do próprio aparelho!

Além da boa qualidade das respostas dadas pelo aparelho, destacamos as seguintes facilidades operacionais:

– permite retroceder sete pares de jogadas;

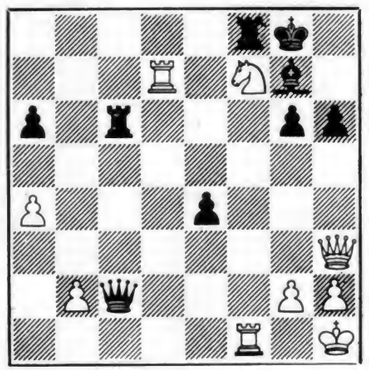
– pensa enquanto aguarda a resposta do adversário;

– informa o que pretende jogar enquanto pensa saltando de uma para outra opção até o final;

– anuncia mate de até sete jogadas e empates nas modalidades afogado, repetição de posição e 50 jogadas sem trocas.

OPORTUNIDADE PERDIDA

Lars-Ake Schneider, MI sueco, ao jogar com Miguel Tahl, ex-campeão mundial, na 13ª rodada da Olimpíada Mundial de 1982, realizada em Lucerna, Suíça, deixou escapar a oportunidade de realizar sua imortal. Ao pousar o rei na casa R1C, para evitar empate após C5C+ e C7B+, repetindo a posição, Tahl percebeu que errara. Só lhe restou ficar impassível e esperar. Seu companheiro de equipe, Gari Kasparov, de pé, atrás de Schneider, viu o que se passava. Outros espectadores também viram, mas ninguém deu alarme para o sueco. Schneider jogou rotineiramente e perdeu.



A posição perdida de Tahl, que seu adversário, o sueco Schneider, não soube aproveitar. Jogam as brancas.

Apresentada a posição ao **Challenger 9**, no nível 4 (torneio), este encontrou a jogada chave da combinação ganhadora em dois minutos e poucos segundos: 1 – DxP!!, demolindo as defesas negra. 1 – ... P4C. Se *1 – ... BxD?,* viria *2 – CxB + R1T; 3 – TxT++.* Se *1 – DxPCD?; 2 – D8T+! BxD; 3 – C6T++.* 2 – DxPC Dx PCD; 3 – C6T+! R2T; 4 – C5B TxC. Se *4 – ...T3CR; 5 – D5T+R1C; 6 – C7R++* 5 – DxT+R3T; 6 – D4B+, seguido de DxP. Espetacular esta análise do **Challenger 9.** Outras subvariantes poderiam ser questionadas ao micro nesta posição rica de variações/combinações. O resultado final seria o mesmo.

O LEITOR ESCREVE

D. C. dos Santos, apaixonado enxadrista do Rio, em gratificante carta sugere que sejam publicadas, ocasionalmente, opiniões dos leitores a respeito de suas experiências com micros.

Meus aplausos pela idéia. Passando das palavras aos fatos, fiz o seguinte resumo de sua carta, a contragosto, pois o material enviado é excelente.

O Sr. Santos tem o micro brasileiro TK82-C e o programa TKADREZ, específico para este micro, com seis níveis de força. Ele declarou ter desistido de jogar nos níveis 5 e 6 em virtude da extrema demora do micro em responder às jogadas. Quanto aos níveis 1, 2 e 3, considera-os para principiantes. O rating do Sr. Santos é por ele avaliado entre 1800 e 1900, depois de comparar seus resultados com seus adversários do Posto 6, no bairro carioca de Copacabana. Ele costuma jogar simultâneas, i. é. enquanto joga, lê, e tem derrotado o pobre computador na proporção de 5 para 1.

Ele declara-se ainda satisfeito por poder jogar com o micro nos dias de chuva, o que não lhe permite enfrentar seus companheiros do Posto 6. O ponto fraco, informa-nos o Sr. Santos, parece ser o das promoções. Só faz dama. Além disso, é impossível manter na memória do aparelho o programa após desligá-lo da corrente.

Finalizando, o Sr. Santos escreve: "No balanço geral, o programa me agrada. Será de grande valia para ensinar meus filhos a jogar Xadrez, além de, vez por outra, servir-me de passatempo".

Tipos, características, vantagens, desvantagens, pontos comuns, aplicações e um resumo de algumas linguagens de programação pouco conhecidas no Brasil.

As linguagens de programação

Tudo o que o computador faz — cálculos, comparações, manipulação de números, caracteres etc. — baseia-se no reconhecimento, pela máquina, de dois estados de corrente elétrica: ligado ou desligado. Esse processo é conseguido pela utilização de dispositivos eletrônicos, geralmente chamados de portas (gates), que somente podem estar abertos ou fechados. À porta aberta, ou ligada, atribui-se o significado de **1**, e à fechada, isto é, desligada, confere-se o valor **0**. Como o microprocessador possui milhares dessas portas, é possível criar unidades maiores de significação pela manipulação de grupos dessas unidades básicas.

Esses **1**s e **0**s são denominados **dígitos binários** (donde o nome **bit,** do Inglês **bi**nary digi**t**) e os grupos desses dígitos recebem o nome de **código binário.** Esse código é o que se utiliza na **linguagem de máquina.** Um número binário em linguagem de máquina representa uma instrução que o computador pode reconhecer e executar. A instrução representada por esse número é determinada pela arquitetura eletrônica do microprocessador: cada um desses dispositivos (8080, Z80, 8085, 6502, 6800 etc.) é construído para interpretar esse número de uma determinada maneira.

OS NÍVEIS DE LINGUAGEM

Se para o computador é fácil entender o código binário, para o homem isso é extremamente difícil e cansativo, uma vez que não estamos familiarizados com a complicada notação binária, de base 2. Para facilitar a comunicação homem-computador, simplificando a tarefa de programar e tornando mais inteligíveis as respostas da máquina, procurou-se modificar a instrução binária original de maneira a que cada uma delas fizéssemos equivaler uma abreviação da palavra que descrevesse a sua função. A tal abreviação chamamos de mnemônico. Como o computador é incapaz de entender esse código simbólico, criou-se o **programa montador** (Assembler), destinado a converter os mnemônicos em linguagem de máquina.

Com o Assembler, as coisas ficaram mais fáceis, mas, a exemplo do código binário, os programas ainda eram muito complexos, exigindo do programador um conhecimento profundo do funcionamento interno do computador. Essas linguagens ficaram posteriormente conhecidas como de "baixo nível", por se situarem muito próximas do nível de hardware.

O advento do FORTRAN, na década de 50, deu origem a um outro tipo de linguagens, denominadas de "alto nível", por estarem mais próximas da linguagem humana do que da máquina. Essas linguagens podem ser compreendidas por nós com muito maior facilidade, pois seus símbolos utilizam palavras em Inglês, grupos de números, palavras especiais e notação matemática padrão.

A exemplo do Assembler, as linguagens de alto nível têm que ser convertidas em linguagem de máquina para serem executadas pelo computador. Isso é feito por um programa específico chamado **tradutor,** que pode ser tanto um **compilador** quanto um **interpretador.** Qualquer linguagem de alto nível pode ser compilada ou interpretada e, dependendo da área de aplicação pretendida, elas tendem a ser implementadas mais de um modo do que de outro.

O compilador pega o programa e o traduz inteiro para linguagem de máquina. A partir daí, numa etapa posterior, essa versão em linguagem de máquina é executada pelo computador. Já o interpretador pode funcionar de duas maneiras: na primeira delas, chamada forma de ordenação em "calculator mode", o interpretador age sobre o programa executando as operações à medida em que as lê, linha por linha. A outra forma de ordenação é aquela em que o interpretador converte as instruções em código intermediário (que não é o código binário) para posterior execução.

Tanto os compiladores quanto os interpretadores têm suas vantagens e desvantagens. Os programas compilados, por exemplo, não podem ser executados — e, portanto, testados — enquanto não forem totalmente convertidos para linguagem de máquina. Assim, é mais difícil detectar erros e corrigi-los. A compilação também consome um tempo bastante longo e ocupa um espaço maior de memória no sistema. Em compensação, os programas compilados são executados muito mais rapidamente (cerca de 15 vezes) do que os interpretados.

A interação com o usuário é a principal vantagem dos programas interpretados, o que significa que o programador pode experimentar diferentes comandos e instruções e descobrir, quase que imediatamente, se eles funcionam. Algumas linguagens utilizam, ainda, modalidades mistas de tradução, interpretando inicialmente os programas e posteriormente compilando-os para execução.

TIPOS E CARACTERÍSTICAS

As principais linguagens de alto nível, como o FORTRAN e o COBOL (e outras que as seguiram, como o ALGOL, BASIC e PL/1) apresentam grande portabilidade, isto é, seus programas podem ser executados em qualquer computador que disponha de um tradutor para a linguagem em que foram escritos.

Esse tipo de linguagem diminui a necessidade de se ter um conhecimento detalhado da máquina em si, permitindo ao programador escrever rotinas capazes de solucionar qualquer problema que ele possa definir. Também reduz o tempo de programação e, como os programas são escritos de maneira mais próxima à linguagem humana, sua documentação e compreensão tornam-se melhores.

Dentro dessa categoria incluem-se as **linguagens conversacionais** que, em sua maioria, originaram-se dos grandes sistemas computacionais de time-sharing (que permitem a vários usuários executarem programas concorrentemente num mesmo computador e interagirem com os programas durante a sua execução). Destinadas ao uso via terminais, numa base pessoal e direta, essas linguagens (que têm no BASIC uma de suas principais representantes) são altamente interativas, permitindo ao programador/usuário receber resposta imediata aos dados e instruções que digita.

Com o passar do tempo, porém, percebeu-se que certos tipos de problemas (ou cálculos) ocorrem tão frequentemente nas áreas científica e comercial, que seria útil desenvolver linguagens específicas para tipos particulares de aplicações: as **linguagens orientadas para problemas.**

O usuário, nesse caso, não precisa ser um programador, mas sim um especialista na área de problema em questão, posto que todos os processos para resolver esses problemas encontram-se embutidos na linguagem de programação. A linguagem ICES (**I**ntegrated **C**ivil **E**ngineering **S**ystem), por exemplo, foi projetada para solucionar problemas específicos de Engenharia Civil. Nela, o engenheiro tem somente que entrar com certos tipos de dados para que o programa faça todos os cálculos necessários, digamos, à construção de um sólido muro de concreto.

Por serem muito especializadas, essas linguagens são pouco flexíveis em termos do seu espectro de aplicação. Também são muito dependentes da máquina para a qual foram originalmente desenvolvidas, o que prejudica a sua portabilidade para outros sistemas. Em compensação, oferecem ao usuário maior eficiência de execução e facilidade de uso para as aplicações às quais se destinam.

Há um grande número de linguagens orientadas para problemas, mas poucas destinam-se a uso em computadores pequenos ou pessoais. Uma delas, contudo, está começando a aparecer em sistemas pessoais, sobretudo em aplicações de âmbito comercial: o RPG (**R**eport **P**rogram **G**enerator) que, como diz o nome, tem o seu forte na geração de relatórios.

STRINGS E LISTAS

Outra modalidade de linguagens são as **orientadas para o processamento de strings e listas,** destinadas especificamente à manipulação de dados (sobretudo não numéricos) onde o comprimento e a estrutura alteram-se consideravelmente durante o processamento de um problema. Algumas linguagens desse tipo lidam somente com strings de caracteres.

Entre as principais aplicações dessas linguagens — que incluem, entre outras, o LISP e o LOGO — podemos destacar a recuperação de informações, comprovação de teoremas, processamento de imagens, geração de padrões, manipulação algébrica e análise linguística.

As linguagens podem ainda ser orientadas para uma série de outras finalidades, como crítica de dados, confecção de sistemas operacionais etc.

Outra característica importante a considerar a respeito de uma linguagem é a capacidade que ela tem, ou não, de permitir ao usuário codificar seus programas de forma estruturada.

A estruturação de um programa é uma técnica que tem dupla finalidade. Em primeiro lugar, organiza o programa em módulos que se interrelacionam em tempo de execução. Tal característica permite maior clareza na codificação, o que é muito recomendável para a manutenção de sistemas. A segunda finalidade é que essa disciplina na codificação torna mais rápida a velocidade de execução.

Referências bibliográficas

Para quem deseja ampliar os seus conhecimentos sobre as linguagens de programação (sobretudo as mais recentes), o prof. Michael Stanton, Coordenador de Pós-Graduação do Departamento de Informática da PUC-RJ, fornece algumas sugestões para leitura:

- LEDGARD — **Ada — an Introduction,** Springer-Verlag, 1981;
- KERNIGHAM e RITCHIE — **The C Programming Language,** Prentice-Hall, 1978;
- **Software — Practice and Experience** — a edição de abril de 1981 dessa revista foi dedicada à linguagem EDISON;
- **BYTE** — a edição de agosto de 1980 dessa revista reúne uma série de artigos sobre FORTH;
- **BYTE** — há diversos artigos sobre a linguagem LOGO na edição de agosto de 1982 dessa revista;
- WIRTH — **Programming in MODULA-2,** Springer-Verlag, 1982;
- WIRTH — **Programação Sistemática,** Campus, 1978;
- CLOCKSIN e TARNLUND — **Programming in PROLOG,** Spring-Verlag, 1981;
- ENNALS — **Beginning MICRO-PROLOG,** Ellis Horwood-Heinemann, 1983.

A estas, podemos acrescentar:

- **INTERFACE AGE** — essa revista apresenta, em sua edição de junho de 1981, uma resenha sobre as principais linguagens;
- **KILOBAUD MICROCOMPUTING** — também uma resenha sobre diversas linguagens, na edição de fevereiro de 1982;
- **80 MICROCOMPUTING** — na edição de julho de 1981, artigos sobre COBOL e PILOT, além de um artigo geral sobre outras linguagens.

»

Uma linguagem que permite a estruturação é constituída por *procedures* e contém instruções do tipo **DO...WHILE, FOR...TO, CASE** etc., além de muitas vezes permitir a recursividade. A recursividade é uma técnica que permite a uma sub-rotina chamar a si mesma sem que o seu estado (status) anterior à chamada seja perdido.

O número de linguagens possíveis de existir é virtualmente infinito. De fato, muitas são as já desenvolvidas, sobretudo em universidades ou laboratórios de pesquisa, utilizando diferentes técnicas, visando aplicações diversas e com variáveis níveis de complexidade e aceitação por parte da comunidade usuária.

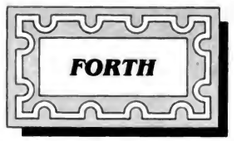
Algumas dessas linguagens são bastante poderosas e, embora ainda a nível de laboratório, já estão chegando ao Brasil, como é o caso da EDISON e da MODULA 2. Derivadas do Pascal Concorrente, e desenvolvidas pelo mesmo autor do Pascal, Niklaus Wirth, essas linguagens estão sendo aqui implementadas no Centro de Pesquisas da Petrobrás — CENPES e na PUC-RJ (que estão desenvolvendo versões da EDISON para o Z80 e o Intel 8086) e na Unicamp (que está desenvolvendo uma versão da MODULA 2 para o processador Intel 8086).

QUAL A MELHOR?

Qual a melhor linguagem? A resposta a essa indagação vai variar de acordo com a pessoa a quem se pergunte. Cada programador ou fabricante de equipamento vai argumentar, em bases sólidas, que a linguagem que ele utiliza é melhor do que as outras. E dentro de uma mesma linguagem podem existir diferentes versões, frequentemente destinadas a acomodá-la às particularidades do projeto de uma determinada máquina.

A determinação da melhor linguagem vai depender da aplicação que se tenha em vista: é preciso antes definir com clareza o que se quer fazer para, em seguida, escolher entre o repertório de linguagens disponíveis aquela cujas características atendam melhor às exigências do trabalho a ser feito.

Nesta edição, as linguagens de programação mais utilizadas são tratadas em artigos específicos, de maior abrangência. Vamos, a seguir, fazer uma descrição sumária de algumas linguagens menos conhecidas no Brasil.



Originalmente desenvolvida para controle de processo (foi criada pelo astrônomo Charles H. More para controlar o seu telescópio no Observatório de Kitts Peak, EUA), a linguagem FORTH tem conseguido despertar o entusiasmo de muitos programadores, apesar de sua estrutura pouco usual e da dificuldade que enfrentam os que desejam aprendê-la (os seus cálculos, por exemplo, são feitos em Notação Polonesa Invertida).

Ela é às vezes chamada de "linguagem inacabada" porque dá ao programador uma liberdade quase infinita de criar novas **palavras** (termo este que, em FORTH, tem um significado similar ao de função).

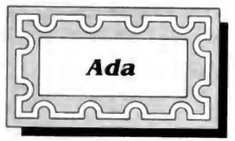
As palavras novas são definidas a partir das velhas, o que reduz muito o custo e o trabalho das sub-rotinas. Escrever um novo programa também é fácil e rápido em FORTH porque, junto com o sistema, vem todo o trabalho que já se fez, como se ele sempre houvesse feito parte da linguagem.

Em FORTH, a maioria das operações se comunica pela utilização de pilhas (a secção da memória onde armazenamos os números na chamada ordem last-in-first-out, ou seja, os últimos a entrarem serão os primeiros a saírem). Todas as linguagens utilizam a pilha, mas sua operação é geralmente incorporada à própria linguagem. Isso não ocorre em FORTH, onde o programador controla a pilha diretamente.

Os cálculos matemáticos não são o forte dessa linguagem, mas ela contorna essa dificuldade através da possibilidade de ligar-se a sub-rotinas em outras linguagens para efetuar o cálculo pesado.

FORTH é uma linguagem estruturada que não tem **GOTOs** ou labels para declarações. Enquanto a maioria das linguagens de programação produz ou um código de máquina para execução direta pela UCP (FORTRAN e PL/M, por exemplo) ou um código interpretador (BASIC, Pascal e LISP), FORTH produz um código modulado. Isso significa que os programas em FORTH são construídos a partir de umas poucas sub-rotinas interconectadas por uma série de **CALLs** para executar a tarefa maior. Esta entidade, por sua vez, é chamada por uma rotina maior e conectada a outras para formar uma entidade ainda maior. A modulação não é uma exclusividade de FORTH, mas é somente nessa linguagem que todas as versões utilizam-na normalmente.

Por ser uma linguagem interpretada, posteriormente compilada em código de máquina, FORTH é muito mais rápida (cerca de 10 vezes) que o BASIC e exige pouco espaço de memória. Além disso, é de baixo custo: há versões para praticamente todos os microcomputadores e mesmo as versões destinadas a aplicações comerciais são baratas e oferecem uma série de facilidades.



O COBOL tornou-se a linguagem comercial padrão porque as agências governamentais dos Estados Unidos assim o exigiram na década de 80.

Agora, o Departamento de Defesa norte-americano decidiu que precisa de uma nova linguagem para coordenar as suas necessidades de aplicação no Exército, na Marinha e na Força Aérea.

O processo de desenvolvimento dessa nova linguagem começou em 1975 com a solicitação de sugestões às três Armas, à indústria e à comunidade acadêmica. Paralelamente a isso, empreendeu-se um exaustivo estudo das linguagens existentes para determinar se alguma delas atenderia às especificações exigidas por essa linguagem. Recomendou-se que o Pascal, o ALGOL 68 ou o PL/1 fosse utilizado como o ponto de partida do projeto. O Pascal foi o escolhido e a nova linguagem foi desenvolvida a partir dele. Desde então, a linguagem Ada tem sido submetida a uma variedade de testes e têm-se desenvolvido compiladores para diversos sistemas de grande porte.

O nome Ada é uma homenagem à primeira programadora de computador, Lady Ada Augusta Byron, Condessa Lovelace, filha do poeta Lord Byron.

O Dr. Kenneth Bowles, líder do projeto que desenvolveu o Pascal UCSD, deixou a Universidade da Califórnia para implementar a Ada em alguns microcomputadores avançados que utilizam os microprocessadores da Western Digital e o chip Motorola 68000. Outras versões deverão surgir. Esse tipo de suporte, aliado ao fato de que a Ada será de uso obrigatório pelo Departamento de Defesa e todos os seus contratados por volta de 1985, indicam que essa linguagem desempenhará um importante papel no futuro.

Ada parece com o Pascal. Ela tem uma parte declarativa e outra de diretivas operacionais. Exige muita digitação porque todos os identificadores têm que ser declarados e os seus atributos especificados.

As duas principais estruturas de controle são as declarações condicionais (que selecionam ações alternativas) e declarações de loop (que especificam a repetição de uma ação).

Ada utiliza muitos tipos de funções e subprogramas denominados *procedures*. Além disso, tem dois tipos de módulos chamados **packages** e **tasks.**

Os **packages** são utilizados para definir coleções de recursos logicamente relacionados para uso em computação.

Tasks são jobs separados que são executados ao mesmo tempo, tanto num ambiente de time-sharing como num sistema de processamento distribuído. O termo coletivo para isso é multitasking, e Ada foi projetada para estabelecer e executar esses jobs como parte do programa. Coisas semelhantes podem ser feitas pelo Pascal, C e PL/1, mas Ada é a primeira linguagem criada para gerenciar as complexas atividades computacionais que utilizam múltiplos processadores e recursos de memória quase ilimitados.

PILOT

Desenvolvida pela Universidade de São Francisco, a PILOT — **P**rogrammed **I**nquiry, **L**earning **O**r **T**eaching foi a primeira linguagem dedicada à instrução assistida por computador, tendo sido implementada nos mais diferentes computadores, dos de maior porte ao mais simples dos micros. Essa linguagem interativa permite que uma pessoa sem conhecimentos prévios de computação possa desenvolver e testar programas conversacionais para o ensino, uma vez que a sua estrutura e sintaxe são fáceis de explicar a um estudante.

Com o auxílio da PILOT o professor pode fornecer ao aluno um trecho de leitura, dar-lhe tempo para estudá-lo e, em seguida, formular questões de múltipla escolha sobre o assunto. O programa pode analisar a resposta e fornecer conselhos ou comentários baseados nessa resposta. O computador também pode introduzir um problema de matemática e mostrar a solução passo a passo ou, ainda, dar ao estudante a oportunidade de descobrir a resposta em quantos passos ele quiser, auxiliado por dicas do computador.

As instruções do PILOT dividem-se nas seguintes categorias:

1. Instruções de núcleo (core instructions) — Essas funções básicas são instruções de uma única letra e se constituem em padrão para todas as versões do PILOT. Graças a isso, os programas são portáveis de uma máquina para a outra. Essas instruções são:

T: TYPE (inclui Y: e N:)	U: USE
A: ACCEPT	E: END
M: MATCH	C: COMPUTE
J: JUMP	R: REMARK

Há também instruções multipalavra denominadas **Keywords,** que foram acrescentadas à PILOT para aplicações específicas e que não constam de todas as versões.

2. Instruções de cursor e vídeo para determinar onde o texto vai aparecer na tela.

3. Instruções que estabelecem vários tipos de parâmetros relacionados com o computador, tais como portas de saída, velocidade de exibição ou localizações de memória.

4. Instruções do sistema de arquivo, relacionadas ao armazenamento e recuperação de programas e dados.

LISP

Linguagem de pesquisa sobre a Inteligência Artificial, LISP — **LIS**t **P**rocessing baseou-se no trabalho de John McCarthy sobre linguagens não numéricas de computadores, publicado em 1960. Foi implementada no MIT — Massachussets Institute of Technology e está descrita no "LISP 1.5 Programmer's Manual".

LISP é uma linguagem não-matemática, composta de palavras, como qualquer língua humana. Em LISP há dois tipos de palavras: **átomos** e **listas.** Os átomos são a entidade básica dessa linguagem. Qualquer combinação de caracteres do alfabeto com qualquer um dos dez dígitos (de 0 a 9) é um **átomo** desde que comece por uma letra. A **lista** é o segundo tipo de palavra em LISP, e é construída a partir de átomos e de outras listas. Uma lista consiste de um parêntese esquerdo seguido por qualquer número de átomos e listas e terminando com o parêntese direito.

Quando uma lista é formada, as células de armazenamento necessárias são tiradas de uma lista de células disponíveis denominada lista de armazenamento livre. Em LISP, uma sub-rotina pode ser considerada como uma ferramenta que define a função num sentido matemático, isto é, mapeia conjuntos de valores de entrada em conjuntos de valores de saída. Em LISP, essa função é expressa numa notação que apresenta a sua natureza funcional mais explicitamente do que normalmente é feito como uma sequência de instruções.

Várias expressões em LISP incluem expressões condicionais, que testam condições e operações, de acordo com o resultado do teste.

A linguagem tem funções variáveis e operadores aritméticos, mas parece estranha aos programadores de BASIC porque todas as operações aritméticas são em Notação Polonesa Invertida. Uma sentença de LISP parece com uma lista, mas carrega significado e é, na verdade, um pequeno programa. Todas as funções LISP têm um valor único e um programa que consiste de funções aplicadas aos argumentos. A linguagem LISP tem muitas funções embutidas, e o programador pode criar funções à vontade.

APL

APL (**A** **P**rogramming **L**anguage) foi desenvolvido nos EUA por Kenneth Iverson e utilizado pela primeira vez num computador IBM em 1960. Linguagem orientada para computação — ao mesmo tempo em que apresenta características de solução de problemas e de uso conversacional — o APL revelou-se tão claro e conciso que a IBM decidiu utilizá-lo em toda a sua série 360.

Desde então, o APL tem sido usado em computadores grandes, notadamente em ambientes time-sharing. Agora, com a possibilidade de transportá-lo para sistemas pequenos mas poderosos, como o TRS-80 e o Apple, cresce bastante o número de pessoas que poderão ter acesso a essa poderosa linguagem de uso geral.

Por ser uma linguagem interpretada, o APL tem o que se chama de modo de execução imediato, isto é, pode-se digitar expressões e alcançar um resultado sem ter que escrever o programa inteiro, ao contrário de outras linguagens em que primeiro é necessário passar por uma ***procedure*** para reservar área de memória, definir tipos de dados e coisas desse tipo.

Seus operadores executam ações que requereriam dezenas de declarações em outras linguagens, o que permite ao iniciante programar funções simples, literalmente em minutos, sem a necessidade de conhecer o funcionamento interno dos computadores e outras coisas relativas aos dispositivos de armazenamento, exigências de memória etc.

O poder do APL baseia-se na utilização de matrizes como estruturas de dados e um conjunto de operadores de invejável extensão para manipulá-las.

Todos os operadores que agem sobre escalares existem tanto no modo simples como no dual. Por exemplo, o operador que produz o maior de dois valores — se aplicado a um valor único — retorna o menor no inteiro que é maior do que ou igual ao argumento.

Há um operador que produz o fatorial de um argumento único ou o coeficiente binomial de dois argumentos. Outro operador funciona como gerador de números aleatórios. Todos os operadores aplicam-se às matrizes sem mudança, desde que as dimensões sejam compatíveis.

PL/1

PL/1 — **P**rogramming **L**anguage One é uma linguagem polivalente projetada para solucionar aplicações tanto comerciais quanto científicas. Ela incorpora as vantagens tanto do FORTRAN quanto do COBOL. É similar ao FORTRAN por suas declarações simples e concisas e ao COBOL por sua habilidade de manipular arquivos agrupados e facilmente incluir e extrair registros de arquivos.

Utiliza blocos básicos de construção denominados ***procedures*** (grupos de instruções que realizam uma determinada função). As ***procedures*** raramente utilizadas podem ser mantidas em dispositivos auxiliares de armazenamento e chamadas para a memória principal somente quando necessário (são os chamados ***overlays***. Uma ***procedure*** pode estar contida em outra.

PL/1 pode manipular cadeias de dados, que podem consistir tanto em cadeias de caracteres alfanuméricos quanto em cadeias de bits. Essa característica é importante. Sem ela, os programadores teriam que utilizar linguagens Assembler para tais problemas. Com o PL/1 o programador pode também descrever os dados em termos de matrizes e estruturas. As primeiras constituem-se numa coleção de dados do mesmo tipo e com características similares, enquanto que as segundas contêm caracteres misturados com campos de dados de diferentes tamanhos. O PL/1 também utiliza ***labels,*** o que lhe permite adaptar-se a qualquer nível de detalhe e legibilidade.

PL/M é uma versão abreviada do PL/1, projetada para sistemas baseados em microcomputador, de limitada potência e capacidade de memória. Foi desenvolvida inicialmente para a Intel Corp., como linguagem exclusiva para os seus dispositivos 8080 e seus sucessores. Foi rapidamente seguida por outras variações, incluindo a MPL e a PL/W, para o Motorola 6800, a SMPL, para as séries IMP e as famílias 89000, da National Semiconductor, a PLuS, para o Signetic 2650, e a PL/Z, para os microprocessadores Z-8, Z-80 e Z-8000, da Zilog.

Há ainda uma versão brasileira dessa linguagem, a LPS, desenvolvida pela Cobra. »

C

Projetada nos Laboratórios Bell, EUA, para operar sobre o poderoso sistema operacional UNIX, C é uma linguagem estruturada, com alguma semelhança com o Pascal. Este, no entanto, utiliza tanto funções quanto *procedures,* ao passo que a Linguagem C alcança a modularidade somente através do uso de funções. Ela constrói toda a estrutura do programa através do uso de funções, a ponto de não ter as declarações **PRINT** ou **READ** e a entrada e a saída também são feitas através do uso de funções. Tem **IF-THEN-ELSE, WHILE LOOPS** variáveis globais e locais e tipos de dados (data types), ponteiros e matrizes. A exemplo do Pascal, pode substituir declarações simples por declarações compostas para promover o fluxo do programa.

C é uma linguagem compilada na qual os programas são compostos com o auxílio de um editor e, a seguir, compilados em versões de código de máquina para serem executados. Essa linguagem não tem estrutura própria de E/S, utilizando a E/S de qualquer sistema operacional no qual estiver implementada.

Um programa C é um conjunto de funções. A habilidade do programador para criar suas próprias funções de acordo com as suas necessidades fazem de C uma linguagem de incomum flexibilidade. Não há linhas numeradas em C. O programa começa com o nome da função, seguido de um colchete para começar a definição da função. Esta definição consiste de declarações compostas entre dois colchetes. As declarações são encadeadas em qualquer nível requerido e tratadas como instruções simples. Há bibliotecas de funções-padrão, funções essas previamente definidas pelo usuário. Todas elas podem ser chamadas para uso no programa. Pode haver variáveis tanto globais como locais, podendo ainda haver argumentos para as funções. Há também expressões que podem ser utilizadas para calcular e armazenar dados. C pode chamar rotinas em linguagem de máquina, bem como qualquer das bibliotecas de funções específicas ou padronizadas.

Apesar de haver sido criada para o sistema operacional UNIX, a linguagem C foi transportada para rodar em outros sistemas operacionais, tais como o CP/M e sistemas similares ao UNIX.

PROLOG

PROLOG (**PRO**gramming in **LOG**ic) é uma linguagem de programação especialmente voltada para a manipulação do conhecimento. O programa PROLOG consiste numa série de fatos sobre determinado assunto, com base nos quais o sistema tentará responder às perguntas que lhe faremos.

Esses fatos são expressos de maneira bastante concisa. O fato, por exemplo, de que João é o pai de José, será expresso pelo seguinte programa on-line PROLOG:

```
pai(joão,josé).
```

A **relação,** pai, aparece primeiro, seguida pelos **argumentos** (entre parênteses) aos quais a relação se aplica. Como regra, o **sujeito** da relação é o primeiro argumento, enquanto que o **objeto** é o segundo argumento.

Essa estrutura de dados é denominada **termo,** e cada termo pode ter qualquer número de argumentos.

A concisão da linguagem PROLOG também se manifesta na forma como são feitas as **asserções** (nas quais informamos os fatos ao sistema) e as **questões** (nas quais fazemos perguntas sobre os fatos informados), que é basicamente a mesma. A única diferença é que a asserção sempre termina em ponto, enquanto que a questão termina em interrogação. Se, por exemplo, queremos saber se João é o pai de José, perguntamos:

```
pai(joão,josé)?
```

Podemos ainda fazer perguntas mais complexas, tais como quem é o pai de José:

```
pai(%quem,josé)?
```

O "%" que antecede "quem" é uma variável. Quando uma pergunta contém uma variável, PROLOG tenta atribuir um valor correto a essa variável:

```
pai(%quem,josé)?
%quem = joão
```

Se não conseguir encontrar o valor da variável, o sistema responde "não".

A exemplo da maioria das linguagens, não importa o nome que se dê à variável, se "% quem" ou "%x". A grande diferença, contudo, é que em PROLOG a extensão da variável limita-se à declaração na qual ela aparece. Se o mesmo nome de variável aparecer em duas declarações diferentes, não haverá nenhuma ligação entre elas, sendo ambas tratadas como variáveis diferentes.

DEDUÇÕES

PROLOG pode também fazer deduções em cima dos fatos que recebe. Presumamos que fizemos duas asserções, informando, respectivamente, que Mário é o pai de João e que este, por sua vez, é o pai de José:

```
pai(mário,joão).
pai(joão,josé).
```

A partir dessas asserções, podemos deduzir que Mário é avô de José. PROLOG também pode fazer essa dedução se lhe ensinarmos que o avô é o pai do pai, fato esse expresso na seguinte cláusula:

```
avô(%x,%z) ← pai(%x,%y),pai(%y,%z).
```

onde ← significa que "é inferido por" ou "é verdadeiro se". O termo à esquerda da ← é verdadeiro se os termos à direita da ← são verdadeiros.

O termo à esquerda é chamado de **meta (goal)** e os termos à direita são denominados **sub-metas (subgoals).** O goal será verdadeiro se os subgoals forem verdadeiros. A meta também é conhecida como o **termo-objeto (head term)** da declaração.

Observemos que foram utilizadas variáveis na cláusula para formular uma declaração geral sobre o que significa para alguém ser avô de outra pessoa. PROLOG pode utilizar essa declaração geral para responder a perguntas específicas como, por exemplo, se Mário é avô de José:

```
avô(mário,josé)?
```

PROLOG tentará responder colocando "%x = mário" e "%z = josé" na definição de avô:

```
avo(mario,jose) ←
pai(mário,%y),pai(%y,josé).
```

Aqui é afirmado que Mário é o avô de José se Mário for pai de uma pessoa que é pai de José. Pelas duas primeiras asserções, PROLOG sabe que Mário é o pai de João e que João é o pai de José. Por isso, PROLOG responde com um "sim".

Se pedirmos a PROLOG para descobrir duas pessoas em que uma seja avô de outra:

```
avô(%x,%z)?
```

PROLOG responderá:

```
%x = mário, %z = josé
```

COMO FUNCIONA

PROLOG tenta, basicamente, solucionar as **metas** da esquerda para a direita. Para uma dada **meta,** PROLOG tenta descobrir a declaração cujo primeiro termo (o único termo numa asserção; o termo à esquerda da ← numa cláusula) pode ser feito para igualar a **meta.**

Ele tenta, então, solucionar as **sub-metas** dessa declaração. Se a declaração for uma asserção, naturalmente não haverá **sub-metas.** Se as **sub-metas** puderem ser solucionadas, PROLOG parte para a próxima **meta.**

Se uma das **sub-metas** não puder ser resolvida, PROLOG volta e tenta encontrar uma outra declaração cujo **termo-objeto** iguale a **meta.** Se, após examinar todas as declarações, não for possível encontrar nenhuma que iguale a **meta** procurada, PROLOG conclui então que não pode solucionar essa **meta.**

Isso não significa, no entanto, que não há solução para a pergunta original. Se a **meta** sobre a qual PROLOG estiver trabalhando for realmente uma **sub-meta** de uma das **metas** originais, pode haver uma solução alternativa da **meta** original que não envolva a **meta** fracassada. PROLOG vai procurar ainda mais e tentar descobrir uma solução alternativa. Ele só desiste quando não consegue encontrar solução alguma para qualquer das **metas** originais.

Ao contrário do que ocorre com as linguagens orientadas para procedimentos, em PROLOG pode-se determinar se uma declaração é verdadeira pelo exame somente da própria declaração. Uma declaração num programa PROLOG corresponde a uma sub-rotina inteira numa linguagem convencional de programação. Assim, os programas PROLOG são extremamente modulares.

Cada declaração em PROLOG apresenta um fato, e não interessa a ordem na qual informamos os fatos a PROLOG. Isso significa que podemos aumentar a potência de um programa pela simples adição de novas declarações e, na maioria dos casos, nem é necessário modificar as declarações que já se encontram no sistema.

Pesquisa e texto: Ricardo Inojosa

Pi

Não pare seu programa nem perca a memória

GERATRON®

Gerador Eletrônico Portátil de 200 VA

O Gerador Eletrônico GERATRON é a solução definitiva para o problema de falha na rede elétrica. Quando esta faltar, GERATRON continuará alimentando o seu micro como se nada houvesse acontecido. Chame um representante hoje mesmo.

GUARDIAN
Equipamentos Eletrônicos Ltda.
ALTA TECNOLOGIA EM ELETRÔNICA INDUSTRIAL
Rua Dr. Garnier, 579 • Rocha • CEP 20971 - RJ
Tels.: (021) 201-0195, 261-6458 e 281-3295 - Telex (021) 34016 - São Paulo: (011) 270-3175, Brasília: (061) 226-0133, Salvador: (071) 241-0064, Natal: (084) 223-1690, Recife: (081) 221-0142

---

digimark
comercial e técnica em computação ltda.

Distribuidor exclusivo dos produtos

MEMOREX

- Diskettes (8')
- Mini Diskettes (5 1/4")
- Fitas Magnéticas
- Discos Magnéticos
- Fitas Impressoras
- Recuperação e Manutenção de Discos Magnéticos
- K-7 Digital

Rua José Antonio Coelho, 824 - São Paulo - SP
Tel.: 571.1437/549.2651/549.2652

Rua Florêncio de Abreu, 681 - conj. 902 - Ribeirão Preto - SP
Tel.: (016) 625.9256 - 636.5866

Rua Monte Azul, 339 - Campinas - SP
Tel.: (0192) 52.5226

CP-500
Microprocessador V-80. Memória 48 K. Vídeo de 12". Cassete, 178 KB em disco. Interface RS 232 C. Impressora 80/132 Col.

Tudo o que você precisa saber, para ter nas mãos o universo dos computadores, está no "Digit-Hall" da Garson.
Os microcomputadores garantem uma administração ligeira e eficiente.
Você fica sabendo sobre seu estoque, contabilidade, folhas de pagamento, créditos, débitos e até jogos eletrônicos.
Enfim, tudo para dinamisar a sua vida e de sua empresa.
Nos cursos — BASIC — nossos técnicos especializados orientarão em todas as utilidades e uso de cada micro.

HP — 85 A
Vídeo de 5".
Impressora Térmica 32 col. Fita magnética.
16/32 K memória.
Linguagem BASIC.

MICRO

COMPUTADORES É NA GARSON

- **Todas as marcas e modelos em pronta entrega.**
- **À prazo em até 24 meses sem entrada ou leasing.**

MICROENGENHO
Microprocessador 6502. Compatível c/Apple. Vídeo a cores. Memória 16/48 K. Disk 5 1/4 ou K7. Recursos sonoros.

DGT-100
Compatível com TRS-80. Microprocessador Z-80 c/2,5 MHZ. Duas interfaces para (K7) cassete. 16/48K memória. Sistema modular.

Garson digit-hall

Rua Uruguaiana, 5
Shopping Center Rio Sul (Aberta às 22 horas)

O MICRO SOB MÉDIDA

173/83

A partir de agora, você tem 24 meses para pagar o Unitron AP II que você precisa já. Passe na CompuShop.

Faz tempo que você já percebeu que um microcomputador é indispensável no seu escritório e na sua casa. Então agora é a hora de você comprar um em condições especiais.
Por isso a CompuShop está oferecendo a mais comprovada versão do computador pessoal: o Unitron AP II.
O Unitron AP II tem todas as qualidades que fizeram do seu similar americano, o micro de maior sucesso no mundo inteiro.
Mas ele não é apenas um microcomputador. O Unitron AP II é um sistema que inclui toda uma linha de programas aplicativos, inclusive capacidade gráfica de alta resolução a cores.
Na CompuShop você encontra software para processamento de texto, transmissão de dados, planejamento administrativo financeiro e uma gama completa de aplicações.
Na CompuShop você ainda tem: cursos, livros e revistas especializados e assistência técnica permanente.
E, para você comprar já o seu Unitron AP II é que a CompuShop programa os seus pagamentos em até 24 meses sem entrada. Com crédito imediato.
Visite logo a CompuShop.

CompuShop

Rua Dr. Mário Ferraz, 37 CEP 01453 São Paulo SP
Telefones: (011) 210-0187/212-9004
TELEX (011) 36611 BYTE BR

Estacionamento Próprio
Aberta de Segunda a Sexta, das 9 às 7 horas, e aos Sábados das 9 às 2 horas.

Corte aqui
Nome:
Endereço:
Cidade: Estado:
CEP:
Sim, quero receber maiores informações sobre o Unitron AP II e sobre a CompuShop.
CompuShop - Rua Dr. Mário Ferraz, 37 CEP 01453 São Paulo SP

VB 744

Veja como funciona o estado E3, de acordo com os diferentes ciclos de máquina, e como são feitas as interrupções no microprocessador.

O estado E3 e as interrupções

Orson Voerckel Galvão

No número passado examinamos com bastante detalhe o que ocorre nos estados **E1, E2** e, com especial atenção, **Ew.** Vejamos agora o que acontece no estado **E3.**

Basicamente, é neste estado que ocorre qualquer transferência de informação de/para a UCP, uma vez que o endereço do dispositivo externo (memória ou periférico) já está disponível na via de endereços. Se o tipo de ciclo de máquina for um **FETCH,** a informação será tratada como uma instrução; caso contrário, o será como um dado.

Em primeiro lugar, vamos observar o que ocorre quando a informação é uma entrada para o microprocessador. Vimos que, com a subida de **2** no estado **E2,** a informação de Status some da barra de dados do micro. A partir deste momento, a via de dados estará disponível para a informação necessária ao ciclo de máquina, devendo a mesma estar disponível para a UCP antes da subida de **2** no estado **E3.**

Com a subida deste sinal, o dado deverá ficar disponível durante todo o tempo restante do estado **E3.** A partir do momento em que a informação de Status some da via de dados (subida de **2** em **E2**) a UCP coloca o pino **DBIN** em um estado alto, de forma a indicar que pode ser iniciada a transferência de informação do mundo exterior para o micro.

Este pino permanecerá neste nível até a ocorrência de **2** no estado **E3.** Observem que, apesar de não mencionado, podem ocorrer estados **Ew** entre **E2** e **E3.** No entanto, o sinal em **DBIN** permanecerá alto.

Esta brincadeira se repetirá para todos os ciclos **FETCH,** leitura de memória/periférico e leitura de pilha. Na figura 1 vemos o diagrama de estado desta operação.

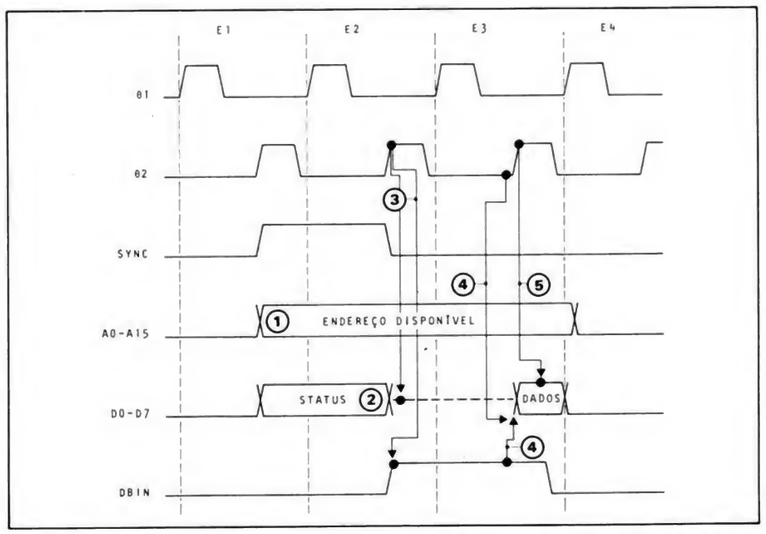


Figura 1

① O endereço do dispositivo de memória/periférico é colocado na via de endereços.
② O Status identifica o ciclo de máquina como um ciclo de leitura.
③ Com a subida de **2** *em* **E2,** *a via de dados torna-se disponível para informação e* **DBIN** *sobe avisando o exterior que pode ser iniciada a operação de entrada.*
④ Os dados terão que estar na via até no máximo o início da subida de **2** *em* **E3.**
⑤ O dado é transferido para o interior do microprocessador.

No caso de saída de dados da UCP, as coisas se processam de forma mais simples. Assim que some o Status da linha de dados, a informação de saída é colocada na via de dados. Durante **1** do estado **E3,** o micro coloca o pino $\overline{WR}$ no estado baixo (lembre-se que seu estado ativo é o baixo), de forma a indicar uma operação de saída. Tanto o sinal WR como a informação contida na via de dados permanecem ativos durante todo o resto do ciclo de máquina. Notem ainda que se ocorrer um estado **Ew** entre **E2** e **E3** a operação não se completará, pois WR só baixa ao entrar-se no estado **E3.** A figura 2 mostra como ocorrem estas operações.

Quanto aos estados **E4** e **E5,** não vamos comentá-los por conterem apenas operações internas à UCP, que já falamos em números anteriores.

Bom, até agora deu para levar o nosso papo com certa tranquilidade. Mas a partir deste momento pediria uma atenção um pouquinho maior, pois vou entrar num assunto meio "cabeludo": são as tais interrupções.

Mas afinal, o que elas são? Veja, você está sentado na sua cadeira "futucando" o seu micro, por exemplo, jogando **Invaders.** O programa consiste basicamente na descida dos invasores para "tomar" a sua cidadela. Você deve bater as teclas ←, →, também a de **espaço,** para, respectivamente, mover-se à esquerda, direita e para atirar nos invasores.

Pois então, quando você bate nestas teclas, você está provocando uma interrupção no programa principal, que consta da descida dos invasores, para que seja realizada uma operação correspondente à tecla apertada. Realizada a operação, o controle retorna ao programa principal.

No BASIC você faz isto com a instrução **GOSUB** e depois **RETURN.** No Assembler, você irá fazê-lo por intermédio da instrução **CALL XXXX** (onde **XXXX** é o endereço da rotina de tratamento de instrução) e depois **RET.** Só que a Intel bolou um conjunto de instruções semelhantes ao **CALL,** que não precisam de endereço, pois o mesmo está implícito na instrução.

São as instruções **RSTn** (**RESTART,** onde **n** varia de 0 a 7), que, em número de oito, pressupõem os endereços fixos de memória 0, 8, 16, 24, 32, 40 e 48.

Qual a vantagem? Ora, elas têm apenas um byte de tamanho. »

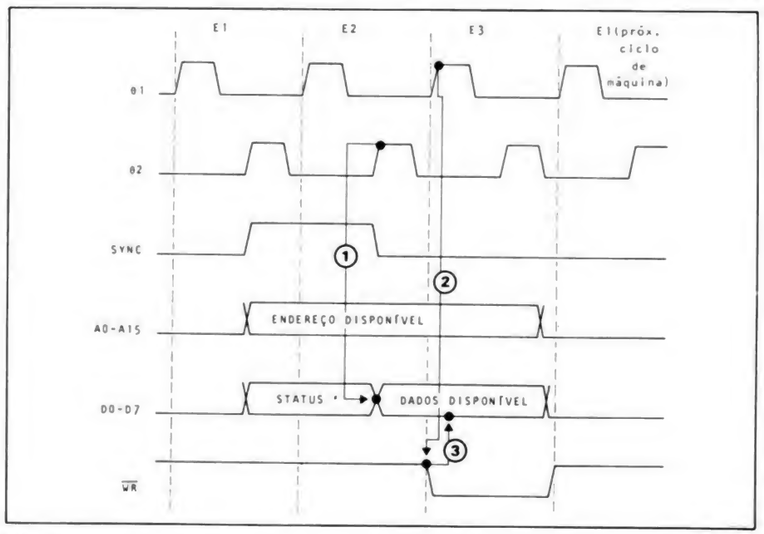


Figura 2

① Com a subida de **2** *em* **E2** *o dado é colocado na via.*
② Com a subida de **1** *em* **E3** *o pino* **WR** *é colocado em seu estado ativo (nível 0), indicando que o dado está disponível para gravação.*
③ O dispositivo de memória/periférico tem todo o estado **E3** *para obter o dado.*

Está bem, só um minuto, e vocês vão perceber o que isto significa. Voltemos à interrupção. De forma semelhante ao nosso programa **Invaders,** a UCP 8080 permite que um periférico a interrompa à qualquer instante durante um ciclo de instrução. Para que isto ocorra, basta que o periférico em questão coloque um nível alto no pino **INT** do microprocessador. Notem bem, isto pode ser feito a qualquer instante. E ainda, uma observação: esta não é a única forma de relacionamento entre a UCP e periféricos. Outro método utilizado é fazer com que a UCP saia perguntando quais periféricos necessitam de seus serviços.

Quando o pino **INT** é colocado no seu nível alto, o microprocessador continua a sequência normal do ciclo de instrução em andamento, de forma a garantir a execução da instrução interrompida. Porém, no último estado, do ciclo de instrução, estando o microprocessador habilitado a receber interrupções e estando o pino **INT** no nível alto, processa-se o início do ciclo de máquina de interrupção.

Mas, que estória é esta de habilitação para receber interrupções? Claro que tem que haver uma certa ordem neste esquema todo, pois muitas vezes pode ocorrer uma segunda interrupção e já se estar processando um ciclo de máquina de interrupção de uma primeira interrupção, o que iria nos causar uma tremenda confusão. Por isto, o 8080 tem um pino chamado **INTE (Interrupt Enable),** que, quando em estado alto, indica que a UCP está apta a receber e tratar interrupções.

O sinal deste pino só irá ao nível baixo em duas ocasiões possíveis:

1 – quando se inicia um ciclo de máquina de interrupção;

2 – ao ser executada a instrução **DI,** que permite o controle do pino **INTE** por software.

Satisfeitos? Então vamos em frente. O ciclo de máquina de interrupção é muito parecido com um **FETCH** (tanto que o bit **M1** do Status deste ciclo de máquina é ligado). O conteúdo do contador de programa (endereço da próxima instrução que seria executada) é colocado na via de endereços durante o estado **E1.** Mas no estado **E2,** contrariando a sequência normal do **FETCH** (e de outros ciclos), o conteúdo do contador de programa não é incrementado.

É pára por aí a diferença. A partir deste momento, entrando-se no estado **E3,** o microprocessador espera encontrar uma instrução na via de endereços. Aí é que se vai encontrar a utilidade da instrução **RST.** O periférico que fez a interrupção, ao sentir que esta foi atendida (através do bit **INTA** do Status), coloca esta instrução de um só byte na barra de dados, ao invés disto ser feito pela memória.

Naturalmente que é necessário um circuito lógico adicional para que o banco de memória seja desconectado da via de dados, de modo que seja obtida a instrução gerada pelo periférico e não a instrução endereçada pelo conteúdo da via de endereços.

A partir do momento em que é obtida a informação na via de dados (a instrução **RSTn**), encerra-se o estado **E3** e a instrução é executada normalmente. Acompanhem na figura 3 o desenvolvimento deste processo, notando que o pino **INTE** volta ao nível alto no término do ciclo de máquina de interrupção.

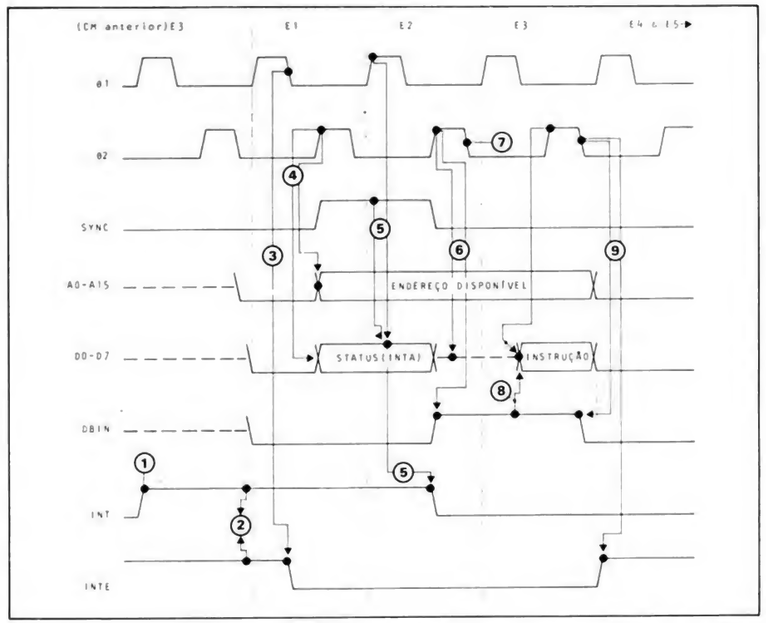


Figura 3

① É solicitada uma interrupção durante o processamento da instrução anterior, que termina normalmente.

② Antes do encerramento do último ciclo de máquina (do ciclo de instrução em execução), o microprocessador "sente" a interrupção e, verificando que está habilitado a atendê-la (pino **INTE** *no nível alto), inicia um ciclo de máquina de interrupção, ao invés de um* **FETCH.**

③ Como primeira providência, é desabilitada qualquer outra interrupção, baixando-se o nível de **INTE.**

④ É colocado o endereço contido no contador de programa na barra de endereços e o Status do microprocessador na barra de dados.

⑤ O periférico que interrompeu, analisando o Status disponível, vê através do bit **INTA** *que a interrupção foi aceita e abaixa o sinal do pino* **INT.**

⑥ Com a subida de **2** *em* **E2,** *a barra de dados é colocada em disponibilidade para receber um dado e o sinal em* **DBIN** *é elevado de forma a indicar que o microprocessador está pronto para receber um dado.*

⑦ O conteúdo do contador de programa não é incrementado.

⑧ Com a subida de **2** *em* **E3,** *o dado (no caso uma instrução* **RST***) tem que ter sido colocado na barra de dados para leitura.*

⑨ Caindo o sinal **2** *em* **E3,** *o dado (a instrução* **RST***) já foi lido, prosseguindo normalmente a execução do ciclo de instrução. O pino* **INTE** *tem o seu nível aumentado, pois já foi iniciada a execução da instrução* **RST** *nos estados* **E4** *e* **E5.**

Agora vamos examinar o ciclo de máquina **HALT.** Após o mesmo ser obtido na memória, depois do ciclo do **FETCH** ele se inicia. Como dissemos anteriormente, o microprocessador entra no estado **Ew** logo após o estado **E2.** Para que se quebre este estado de suspensão existem três formas distintas. Uma delas, já mencionadas, é provocar uma interrupção no microprocessador através da colocação de um sinal alto no pino **INT** (claro que se pressupõe que o pino **INTE** esteja no seu nível alto). Quando isto ocorre, o 8080 entra no estado **E1** de um novo ciclo de máquina de interrupção.

Outra forma de se sair do estado de suspensão será colocando-se um sinal alto no pino **RESET** da UCP. Este pino, quando colocado no nível alto, zera toda a UCP, funcionando no momento em que se liga o micro.

E a terceira forma envolve um tal de pino **HOLD,** que não discutiremos no momento. A sua função está ligada a um certo **DMA** (Acesso Direto à Memória), um tópico que exige um capítulo à parte. Quanto ao **HALT,** vamos apenas dizer que, quando ele recebe um nível alto, o estado **Ew** é encerrado. Mas quando o seu nível baixa novamente, o estado **Ew** continua.

Como vocês já devem ter sentido, o papo começou a ficar meio indigesto... Mas não se preocupem, pois não vamos muito mais adiante. A finalidade desta série é apenas iniciar o leitor no mundo do microprocessador, de forma que ele tenha um trampolim para iniciar uma pesquisa mais profunda sobre os tópicos abordados.

No próximo número vamos bater um papo sobre o **DMA,** o pino **HOLD** e fazer uma pequena sinopse de tudo o que foi apresentado até o momento. E se sentirem que "tá ruço", não se avexem: ESCREVAM!

Orson Voerckel Galvão é Analista de Sistemas da Petrobrás Distribuidora S. A. e Assessor Técnico de MICRO SISTEMAS.
Orson foi o autor do Curso de BASIC publicado por MICRO SISTEMAS, do nº 2 ao 9 da revista.

Saiba o que são e que utilidade têm os montadores, compiladores e interpretadores.

Os programas tradutores

Paulo Roberto Campos Ducap

Com o objetivo de tornar mais fácil a comunicação entre o homem e a máquina, foram desenvolvidos conjuntos de códigos de melhor compreensão para o homem, ou seja, as linguagens.

Inicialmente, para substituir o código numérico digital do sistema binário utilizado pelo computador (também chamado ***linguagem de máquina)*** foi desenvolvida uma linguagem simbólica, à base de mnemônicos, mais próxima à linguagem do homem — apesar de redundar quase que imediatamente na linguagem de máquina. Cada instrução simbólica corresponde praticamente a uma instrução de máquina.

Por exemplo, esta instrução simbólica (ou Assembler):

```
MOV A,B
```

pode ter o seguinte significado: mover o conteúdo da posição **A** de memória para a posição **B.**

Continuando com o esforço de tornar a linguagem de programação mais simples e compreensível, foram desenvolvidas as linguagens de alto nível, já bem acessíveis e de fácil compreensão para o programador sem grandes realizações. Cada instrução em linguagem de alto nível corresponde a uma ou mais instruções em linguagem de máquina que, por sua vez, compõem uma *rotina* para executar determinada função.

A instrução mencionada anteriormente, por exemplo, de acordo com a linguagem utilizada, poderia ser reescrita da seguinte forma:

```
MOVE A TO B

ou

B=A
```

Tanto as linguagens de alto nível — como FORTRAN, COBOL, BASIC ou PL/1 — quanto as simbólicas devem ser "traduzidas" para a linguagem de "trabalho" do computador. Em outras palavras, um programa escrito em BASIC, por exemplo, será sempre traduzido para a linguagem de máquina para que suas instruções possam ser processadas. Esta é a função dos ***programas tradutores.***

Um programa ainda não traduzido é denominado ***programa fonte,*** enquanto que o resultado da tradução em código de máquina é denominado ***programa objeto.***

PROGRAMAS TRADUTORES

Os programas tradutores podem ser divididos em três famílias:

- **Montadores** — traduzem o programa fonte em linguagem simbólica para programa objeto e geram informações úteis para a carga do objeto na memória;
- **Compiladores** — convertem o programa fonte, como um todo, escrito em linguagem de alto nível, para linguagem de máquina;
- **Interpretadores** — traduzem e executam cada instrução do programa fonte dentro da sequência lógica do programa.

Os fabricantes dotam seus computadores de, pelo menos, um montador simbólico e um tradutor de linguagem de alto nível. Outros tradutores podem ser adquiridos junto ao próprio fabricante ou casas de software especializadas.

COMPILADORES

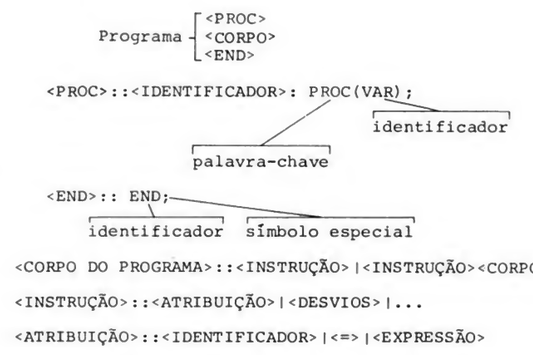
Pode-se definir um compilador como uma rotina executiva que obtém e combina as necessárias partes de informação com a finalidade de produzir um programa executável (módulo objeto). Trata-se de um programa singular, cujo **output** é outro programa. Da mesma forma que o código objeto que produz, um compilador é específico do computador para o qual é projetado.

Durante o processo de compilação podem ser observadas quatro etapas distintas: reconhecimento de elementos básicos, análise sintática, alocação de memória e geração de um programa objeto.

- ***Reconhecimento de elementos básicos*** — é feita uma análise léxica. Pesquisando sequencialmente, o programa vai identificando os elementos que são delimitados por brancos, símbolos especiais (por exemplo, ; :) ou operadores (por exemplo, =). Os elementos básicos são definidos em três classes:
  - identificadores (variáveis);
  - literais;
  - terminais (palavras-chaves, símbolos especiais).
- ***Análise sintática*** — o conjunto de elementos é visto como um todo e tenta-se dar-lhe um sentido. De acordo com o sentido, será escolhida a rotina que interpretará os elementos.

Nesta análise são utilizadas regras de sintaxe gerais, denominadas **reduções,** que interpretam o sentido de cada conjunto de elementos.

Exemplo:



... e assim por diante.

- ***Alocação de memória*** — para todas as variáveis do programa e literais definidas. É efetuado um cálculo para reserva de espaço em memória, associada ao endereço relativo. Ao final da etapa, será conhecido o espaço necessário para a execução do programa.

Alguns compiladores, nesta fase, podem opcionalmente gerar uma listagem simbólica correspondente ao programa.

- ***Geração do programa objeto*** — nesta fase o programa produz suas saídas: listagem comentada e com indicações de erros do programa fonte e/ou gravação do módulo objeto em memória auxiliar disponível.

INTERPRETADORES

O interpretador é um programa que executa as instruções de um programa fonte, linha a linha. Em outras palavras, o interpretador executará a menor unidade que possua algum significado em termos de processamento dentro da linguagem de programação. O **output** do interpretador é o próprio resultado da execução do programa. Em alguns sistemas que operam em ambiente time-sharing (tempo compartilhado) pode haver a divisão do processo em duas fases: geração de um código intermediário e interpretação deste código. Este método possui, neste ambiente, a vantagem de economizar espaço na memória principal e otimizar a gerência dos processos dos vários usuários. De um modo geral, no entanto, os sistemas utilizam a interpretação direta do código fonte.

Um interpretador possui; basicamente, as seguintes características:

— utiliza uma área de memória (partição) para interpretar a rotina fonte do usuário;
— é voltado para a execução de uma unidade de instrução (linha), formada por uma sequência de comandos e seus argumentos num formato definido, terminada por um código de fim de linha (interpreta estes comandos e argumentos da esquerda para a direita);
— identifica a unidade (linha) através de um rótulo;
— dispõe de facilidades (recursos) para criação, modificação, depuração, armazenamento e execução dos programas das linguagens, através, geralmente, de comandos interativos (de comunicação direta, via terminal);
— permite a execução direta de instruções, ou seja, executa imediatamente linhas de instrução sem rótulo;
— aloca dinamicamente espaço de memória para variáveis não declaradas pelo programador;
— fornece recursos de linguagem, de maneira que os programas utilitários são escritos na própria linguagem.

Durante o desenvolvimento de tradutores de linguagens utilizadas em equipamentos de grande porte em ambientes de multiprogramação, os projetistas preocuparam-se mais com o tempo de execução e a alocação de recursos computacionais do que com o tamanho dos programas. Levando-se em conta estas características, foram desenvolvidos os compiladores para linguagens mais populares, como COBOL, FORTRAN, PL/1 e ALGOL. Esta decisão se justifica por serem os programas compilados mais rápidos e com código final mais eficiente (módulo objeto) que os interpretados (onde a eficiência da execução depende exclusivamente da eficiência do código fonte), embora produza programas maiores.

Outra vantagem é a maior segurança quanto à integridade dos programas armazenados, pelo fato de serem módulos objetos.

Existe, entretanto, a dificuldade de ter-se que reiniciar o processo de compilação e geração de novo código objeto (bastante lento, em alguns compiladores, especialmente quando se trata de microcomputadores) cada vez que é feita uma alteração no programa.

Já os interpretadores possuem a vantagem de permitir um tratamento (criação, alteração e teste) bem mais rápido dos programas, especialmente no que toca a depuração e testes.

Também são mais econômicos em termos de espaço de memória e de armazenamento, já que trabalham com programas menores (fontes). Esta vantagem também é especialmente importante para os micros.

O ideal é tentar, sempre que possível, aproveitar as vantagens dos dois tipos de tradutores, utilizando os interpretadores durante a fase de desenvolvimento e testes de programas e os compiladores para gerar os módulos objetos dos programas em produção.

Paulo Roberto Campos Ducap é Analista de Sistemas da Nabla Engenharia e Processamento de Dados Ltda.

INFORMÁTICA 83 NA PAUTA

Durante reunião realizada recentemente em São Paulo entre representantes da SUCESU-SP e da ABICOMP, ficou decidido que ambas entidades vão organizar o XVI Congresso Nacional de Informática e elaborar conjuntamente seu temário.

Já há algum tempo a ABICOMP reivindica uma participação mais efetiva nos congressos. Durante o XV CNI e II FNI, ano passado no Rio, a entidade chegou a afirmar que se retiraria de ambos eventos se a SUCESU não a "convidasse" a participar do temário de futuros congressos. De acordo com os empresários muitos dos temas discutidos nos painéis e seminários não lhes beneficiavam.

E o Informática 83 está prometendo ser um grande sucesso: seis meses antes de sua realização já é grande a procura de espaços para a exposição na III Feira Internacional de Informática. Mais de 40 empresas já reservaram seus stands: Itaú Tecnologia; Stratus Informática; Approach Serviços Ltda.; Sisco Serviços de Computadores; Logus Computadores Ltda.; Organização Ruf; Compucenter Ltda.; e Brascom Computadores entre outras.

O Informática 83 vai se realizar entre os dias 17 e 23 de outubro, no Parque Anhembi, em São Paulo. Os interessados na reserva de espaços para a III Feira devem procurar a Guazelli Associados, na Rua Manoel da Nóbrega, 800, tel. (011) 285-0711, São Paulo. E as inscrições para o XVI CNI podem ser feitas na Secretaria do Congresso, na Avenida Paulista, 1159/14º andar, cj. 1404/1405, tel. (011) 288-9452.

O MICRO DA ITAUTEC

O I-7000, microcomputador da Itautec, totalmente projetado no Brasil, chegou recentemente no mercado. Antes mesmo de ser lançado oficialmente, o equipamento da Itautec trazia alterações sobre o projeto original: os técnicos da empresa concluíram que o microprocessador 8085A não traria a melhor performance ao I-7000 e, em menos de um mês, modificaram o projeto.

Em vez do microprocessador 8085A, o I-7000 está sendo comercializado com o NSC 800, permitindo, com esta mudança, maior velocidade de processamento, já que o NSC 800 exige menor tempo de resposta.

O I-7000 tem 64 Kb de memória RAM, 4 Kb EPROM (ambos expansíveis até 128 Kb), opera com o Sistema Itautec para Microcomputador (SIM/M, compatível com o CP/M), comercializado em cartucho ou disquete. O micro da Itautec já conta com software desenvolvido pela própria empresa, como o SET – Sistema Emulador de Terminais – que possibilita a ligação do I-7000 a um terminal IBM 3270; o SED – Sistema de Entrada de Dados; e ainda o Redator, para processamento de textos. O I-7000 custa, em sua configuração básica, cerca de Cr$ 1 milhão e 500 mil.

GRAVADOR DE MEMÓRIAS

A empresa paulista Microway Tecnologia Eletrônica Ltda. desenvolveu o gravador de Memórias MW-27, com hardware baseado no microprocessador Z80A, criando maior possibilidade para o usuário ampliar a capacidade de programação de seu equipamento de acordo com as suas necessidades.

Entre inúmeras outras vantagens que o MW-27 permite, algumas são de destacada importância para a performance da programação, tais como: verificação e/ou comparação da memória interna com a externa; inserção ou remoção de dados para correção; leitura e transferência de dados da memória interna; memória interna disponível para substituição da PROM e EPROM externa; movimentação interna de blocos de dados; auto monitoração das operações com mensagens de erro; memória virtual e identificação de memórias por soma de dados.

SPECTRUM

Muito se tem comentado sobre os computadores domésticos coloridos. Todas as revistas estrangeiras trazem um cardápio variado, com grandes nomes como BBC, Spectrum, VIC 20, ORIC-1, LYNX, Jupiter Ace, isso só falando dos pequenos. Nem todos estão disponíveis aos brasileiros, ou pelo menos acessíveis mas, apesar disso, um deles parece já ter sido eleito o preferido: o Spectrum, da Sinclair.

Afinal, o que é o Spectrum? É um computador pessoal, cujos principais atrativos são o baixo custo e o display colorido. Ele é uma versão sofisticada do ZX81, onde certas falhas foram corrigidas e algumas novidades incorporadas.

O teclado é feito de borracha (do tipo do CP-200 e do TK85); cada tecla pode ter até seis funções diferentes (dependendo do modo de operação); o BASIC residente, de 16 Kb, similar ao BASIC do ZX81, ganhou novos comandos como **READ, DATA** e **RESTORE**; e a velocidade de execução dos programas foi fixada em **FAST**, embora o display mostre em **SLOW**.

A velocidade de gravação em cassete foi aumentada e dois novos comandos incorporados: **VERIFY**, que checa se ocorreu falhas na gravação do programa; e **MERGE**, que permite a gravação de sub-rotinas da fita no programa que já está no computador.

O ponto alto, porém, é o display e suas operações. O vídeo é formado por 22 linhas de 32 caracteres, e oito cores podem ser estabelecidas, tanto para o primeiro plano quanto para o fundo (preto, branco, azul, ciã, magenta, amarelo, verde e vermelho), além do brilho, que pode ser normal ou extra. Os caracteres são padrão ASCII, com letras maiúsculas e minúsculas, e com alguns caracteres gráficos. A novidade é que 21 caracteres podem ser definidos pelo usuário, aumentando bastante a variedade de caracteres disponíveis.

O Spectrum possui ainda um gráfico de alta resolução, formado por um matriz de 256 x 192 pontos, o que permite acesso a todos os pontos do display.

Com estas principais características, além de um design elegante e moderno (bem no estilo europeu), o Spectrum prova o avanço e a importância dos pequenos computadores no mercado. Mas há um probleminha que deve ser considerado pelos brasileiros: o Spectrum funciona com o sistema a cores PAL britânico, que não é compatível com o sistema adotado aqui, o PAL-M.

SERVIMEC INAUGURA CENTRO EXPERIMENTAL

Há 27 anos no setor de cursos e consultoria, a Servimec Processamento de Dados inaugurou, em março, o seu Centro Experimental de Informática. Segundo Antonio Barrio Jr., diretor superintendente da empresa, a principal função do CEI é orientar e procurar juntamente com o usuário a melhor solução para seus problemas empresariais, e somente depois efetuar a venda de um computador.

No CEI o usuário encontrará à sua disposição equipamentos e software de vários fabricantes: Polymax, Labo, Scopus, Prológica, Cobra, Dismac, Microdigital. Unitron, Spectrum, toda a linha de calculadoras técnico-científicas da Texas e Hewlett-Packard, além de uma equipe composta por um analista de sistemas, cinco programadores e cinco consultores em informática. No CEI o usuário também encontrará software desenvolvido pela equipe da Servimec.

BASIC EM CASA

A empresa paulista SDI, que atua nas áreas de assessoria para seleção de micros, implantação de equipamentos, revenda de TK82-C e Microengenho e software para equipamentos compatíveis com o Apple e o Sinclair está implantando um método inédito no Brasil para o ensino de BASIC.

A metodologia da SDI, baseada em similar americana, permite que o aluno, de posse de um manual dirigido e utilizando um micro pessoal, possa fazer, tranquilamente, o curso de BASIC na sua própria casa.

SEMINÁRIO DA ABAM

A Associação Brasileira de Administração de Material – ABAM – realizará, de 25 a 28 de abril no Hotel Rio Palace, no Rio de Janeiro, o seminário sobre MRP II – Manufacturing Resource Planning (Planejamento dos Recursos de Manufatura), promovido pela empresa educacional americana Oliver Wight Inc., com palestras do professor e consultor da empresa, Al Stevens.

O seminário, com tradução simultânea, abordará o desenvolvimento e implantação de sistemas para planejamento e controle dos estoques e da produção. O seminário MRP II, que é uma evolução do MRP (Planejamento das Necessidades de Materiais), terá também uma exposição de software para a administração industrial, com a participação das maiores empresas de software do mercado brasileiro, apresentando seus sistemas de planejamento e controle de manufatura.

Os interessados em participar do seminário da ABAM devem entrar em contato com a entidade, na Av. Beira Mar, 406/gr. 207, tel. (021) 220-8490, CEP 20021, Rio de Janeiro-RJ. A taxa de inscrição para os sócios da ABAM é de Cr$ 120 mil, e para os não sócios é de Cr$ 150 mil.

CPD DE MICROS

O Colégio Brasil, em São Paulo, inaugurou recentemente o seu Centro de Processamento de Dados, composto por 22 microcomputadores modelos SID 3000 e 3300. O CPD é para utilização exclusiva dos alunos e vai inclusive funcionar nos finais de semana para aqueles que quiserem praticar nos equipamentos. Como parte do currículo do 2º grau, o Colégio Brasil oferece um curso técnico de processamento de dados, com duração de três anos, cujo objetivo é a formação de profissionais em nível médio. Os alunos aprendem a programar em BASIC e COBOL, e recebem também noções de Assembler e FORTRAN. Segundo o coordenador do CPD, Luiz Carlos Tobaruela, várias empresas que utilizam os micros da SID já entraram em contato com o Colégio Brasil para futuro aproveitamento dos alunos que concluírem o curso.

CONSÓRCIO CIBERNÉTICO

Para quem acredita na sorte e quer comprar um micro a prazo, desde a segunda quinzena de janeiro o Consórcio Nacional Garavelo está comercializando o CP-500 da Prológica. Cada consorciado paga uma prestação inicial de Cr$ 31 mil 993 e mais 35 parcelas de Cr$ 24 mil reajustáveis de acordo com os aumentos do fabricante. Segundo Eudoro Lemos, gerente de vendas do Consórcio, por mês saem invariavelmente dois equipamentos: um por sorteio e o outro por lance. Até o final do ano o Garavelo pretende ter aproximadamente 720 consorciados.

A comercialização do CP-500 pelo Consórcio Garavelo coincidiu com nota distribuída à imprensa pela Prológica, onde a empresa anuncia que quadruplicará seu faturamento, devendo atingir este ano a casa dos Cr$ 24 bilhões. Segundo dados da Prológica, esse faturamento será possível vendendo-se 34% de microcomputadores, 30% de micros pessoais, 15% de impressoras, 13,5% de discos e 6% de serviços.

Assembler

Nascido junto com o computador, o Assembler mantém até hoje a sua utilidade.

A primeira linguagem

Amaury Moraes Jr.

Quando surgiram os primeiros computadores, sentiu-se a necessidade de fazer com que essas máquinas fossem orientadas para diversas aplicações, uma vez que o preço e o tamanho dos computadores da época tornavam totalmente inviável a utilização de um equipamento diferente para cada aplicação.

A partir desse princípio, desenvolveu-se um conjunto de instruções específicas para cada processador, cada uma delas executando ações previamente estabelecidas no projeto do processador.

Como era muito difícil programar diretamente os passos que o processador deveria executar para se obter um determinado resultado, criaram-se as linguagens de programação, que podem ser classificadas em dois grandes grupos.

ALTO E BAIXO NÍVEL

As linguagens de alto nível destinam-se ao desenvolvimento de aplicações rotineiras e apresentam as seguintes características: compatibilidade entre computadores, facilidade na contratação de mão-de-obra, facilidade na elaboração do programa (ficando toda a complexidade do software envolvido transparente ao programador), facilidade na manutenção dos programas e transparência quanto ao Sistema Operacional utilizado. Entre as linguagens de alto nível podemos destacar o COBOL, FORTRAN, BASIC, LISP e APL, entre outras.

As linguagens de baixo nível orientam-se para o desenvolvimento de aplicações específicas, tais como sistemas operacionais, compiladores e intérpretes, utilitários, gerenciadores de banco de dados etc. Nessas linguagens, o programador trabalha praticamente ao nível da linguagem de máquina, conseguindo performances superiores, tanto em velocidade de processamento quanto em memória utilizada. O programa é dependente do hardware existente, não havendo compatibilidade entre equipamentos: um programa em Assembler desenvolvido, por exemplo, para o Apple, não é executável no TRS-80 e vice-versa. Não há muitos profissionais especializados em Assembler no mercado, e a dificuldade na documentação e manutenção dos programas é bem acentuada. Todo Assembler é uma linguagem de baixo nível.

A LINGUAGEM ASSEMBLER

Essa linguagem nasceu junto com a computação e permaneceu até nossos dias.

Cada instrução em um processador é codificada em código de máquina, isto é, um código binário que executa uma determinada ação no microprocessador. O código de máquina é específico para cada microprocessador. Excepcionalmente, alguns micros apresentam compatibilidade entre si, como no caso do Z80, que é compatível a nível de código de máquina com o 8080 (a recíproca, entretanto, não é verdadeira).

Como podemos observar na figura 1, o código de máquina 21H executa a mesma instrução de máquinas nos microprocessadores 8080 e Z80. Entre outros processadores, no entanto, não há nenhuma correspondência para o mesmo código de máquina.

Outro aspecto que normalmente causa muita confusão é a compatibilidade entre equipamentos que utilizam o mesmo processador. Para que haja compatibilidade, não basta que os equipamentos se utilizem do mesmo processador: é necessário também que os endereços de E/S e o mapeamento de memória sejam compatíveis. Assim sendo, apesar do TK82-C e do CP-500 utilizarem o mesmo processador (Z80), não há nenhuma compatibilidade entre eles.

CÓDIGO DE MÁQUINA	FUNÇÃO	MICROPROCESSADOR
21 H	HL ← NN	Z80
21 H	HL ← NN	8080

Figura 1

Programar a nível de código de máquina é extremamente difícil pela necessidade que teríamos de memorizar todos os códigos e as possíveis ações por eles executadas. Visando eliminar essa dificuldade, cada fabricante determina uma abreviação — chamada **mnemônico** — para cada instrução, o que torna muito mais fácil a tarefa de programar. A programação através de mnemônicos chama-se Assembler, enquanto que a transformação do mnemônico em código de máquina tem o nome de Assembly (figura 2).

MNEMÔNICO	CÓDIGO-DE-MÁQUINA	MICROPROCESSADOR
LD HL, 4000H	21 00 40	Z80
LXI H, 4000H	21 00 40	8080

Figura 2

ASSEMBLER VS. ALTO NÍVEL

Uma programação Assembler apresenta certas características próprias que a diferenciam substancialmente das linguagens de alto nível. Essas diferenças básicas são as seguintes:

1 — Nas linguagens de alto nível, as áreas de trabalho são criadas e manipuladas pela própria linguagem, enquanto que na programação Assembler o programador deve criar e manipular as áreas de trabalho (figura 3);

BASIC	ASSEMBLER
A = 5	LD (4000H),5
B = 7	LD (4001H),7
C = A+B	LD A,(4000H)
	LD B,(4001H)
	ADD A,B
	LD (4002H),A

Figura 3

2 — As comparações nas linguagens de alto nível são realizadas em uma única instrução, enquanto que num programa Assembler qualquer comparação é realizada em duas fases: comparação e decisão (figura 4);

BASIC	ASSEMBLER

IFA = B THEN 100	LD A, (4000H)
	LD HL, 4001H
	CP (HL)
	JP Z, 1000H

Figura 4

3 — Enquanto um programa escrito em linguagem de alto nível é fácil de ser analisado, um programa Assembler é uma verdadeira colmeia, já que o programador é obrigado a escrever muitas subrotinas que se entrelaçam, visando reduzir o espaço ocupado pelo programa. Em um equipamento TRS-80 podemos encontrar a seguinte sequência de passos:

```
CALL  01C9H - Subrotina que gera um CLS
CALL  1B49H - Subrotina que gera um NEW
JP    1A19H - Ponto de entrada do BASIC
```

OPÇÃO I	OPÇÃO II
LD HL, 4000H	LD HL, 4000H
LD DE, 5 H	LD DE, FFF4H
OR A	ADD HL, DE
SBC HL, DE	

Figura 5

Para o desenvolvimento de um programa em Assembler é indispensável o uso de um **editor Assembler.** Esse programa nos permite a entrada de um programa com mnemônicos e endereços relativos para, em seguida, executar a montagem do programa em linguagem de máquina, convertendo os mnemônicos em código de máquina e os endereços relativos em endereços absolutos, calculados a partir de um endereço-base definido pela pseudo-instrução **ORG** (ORIGIN). O exemplo da figura 6 mostra a saída de um editor Assembler. A primeira e a segunda linhas se referem ao programa montado em código de máquina, enquanto que as linhas subsequentes referem-se ao programa Assembler digitado através do editor Assembler. Esse programa gera um vídeo com o caráter gráfico **191** presente em todas as posições do vídeo.

POSIÇÃO DE MEMÓRIA	CÓDIGO DE MÁQUINA	Nº LINHA	LABEL	MNEMÔNICO	COMENTÁRIOS
7000		100		ORG 7000H	PONTO DE CARGA DO PROGRAMA
7000	01 00 04	110	START	LD BC,1024	Nº DE BYTES PARA MOVER
7003	21 00 3C	120		LD HL,15360	1ª POSIÇÃO DE VÍDEO
7006	16 BF	130		LDD,191	CARÁTER A SER MOVIDO
7008	72	140	LOOP	LD (HL),D	MOVE CARÁTER PARA O VÍDEO
7009	23	150		INC HL	PRÓXIMA POSIÇÃO DE VÍDEO
700A	0B	160		DEC BC	DECREMENTA CONTADOR DE POSIÇÃO
700B	78	170		LD A,B	MOVE MSB DO CONTADOR PARA A
700C	B1	180		OR C	TESTA SE CONTADOR = 0
700D	20 F9	190		JR NZ,LOOP	SE BC<>0 VÁ PARA LOOP
7000		200		END START	INICIAR O PROGRAMA EM START

Assembler
PROGRAMA MONTADO

Figura 6

4 — O programador Assembler é obrigado também a lançar mão de certos artifícios para otimizar seu programa. Esses recursos, pouco recomendáveis porém necessários, visam reduzir o espaço ocupado ou o tempo de execução de um programa. O tempo de execução da opção II (conforme nos mostra a figura 5), é oito períodos de tempo mais rápido e ocupa dois bytes a menos de memória, justificando o uso de artifícios de programação.

***Amaury Correa de Almeida Moraes Junior** é formado pelo curso de Análise de Sistemas da FASP, tendo feito diversos cursos de aperfeiçoamento nas áreas de eletrônica digital e microprocessadores.*
***Amaury** trabalha como analista na PRODESP, na área de mini/microcomputadores, presta consultoria a empresas para a implantação de sistemas de microcomputadores e é o autor do "Curso de Assembler" que MICRO SISTEMAS está publicando desde o número 17.* ■

BASIC

O BASIC é a linguagem mais usada em microcomputadores no mundo inteiro. Neste artigo, traçamos uma comparação entre os BASICs do TRS-80 Mod. III, Apple II e Sinclair ZX81.

Três faces da mesma linguagem – I

Orson Voerckel Galvão

Já colaborador de MICRO SISTEMAS há mais de um ano, não foram poucas as sugestões que recebi para que escrevesse um artigo no qual fossem comparadas as instruções BASIC dos diversos equipamentos encontrados no mercado.

Bem... Ao contrário do que possa parecer, não é uma tarefa difícil. Mas vou te contar: é um trabalho que exige uma boa paciência de Jó! Mas como o nosso leitor merece, vamos lá. Neste número, que tem como tema central as linguagens de computação, estou não só fazendo um pequeno "dicionário" de instruções BASIC, mas também a descrição sumária da finalidade de cada uma.

Como, no entanto, quase todos os equipamentos mais populares têm como base os micros TRS-80 Mod. III, Apple II e Sinclair ZX81, nada mais natural do que aqui utilizarmos o nome de tais máquinas ao invés dos similares nacionais. Espero, com isso, não estar incorrendo em pecado mortal junto aos mais xenófobos e, a esse respeito, me penitencio publicando na figura 1 a lista dos equipamentos nacionais e respectivos "gurus" estrangeiros.

Como critério de apresentação, adotarei a seguinte norma: primeiro serão apresentadas as instruções encontradas nos três equipamentos. Em seguida, as encontradas em apenas dois deles e, por último, aquelas exclusivas de cada equipamento. Cabe ainda a observação de que aqui não estão incluídas novas instruções porventura implementadas em equipamentos tupiniquins. Vamos, então, às instruções.

• **ABS (X)** – Esta função tem a mesma sintaxe nos três equipamentos. Sua finalidade é fornecer o valor absoluto do número ou expressão **X.**

TRS-80 Mod. III	APPLE II	SINCLAIR ZX81
CP-500	AP II	NE Z8000
Naja	DEL MC 01	TK-82C
DGT-100 (*)	Maxxi	TK-85
D-8000/1/2 (*)	Microengenho	CP-200

(*) totalmente compatíveis com o TRS-80 Mod. I e parcialmente com o TRS-80 Mod. III.

Figura 1 – Tabela de compatibilidade entre os equipamentos nacionais e as linhas TRS-80 Mod. III, Apple II e Sinclair ZX81.

• **AND** – Este é um operador cuja sintaxe é a mesma nos três equipamentos. Este operador fará com que o resultado de uma expressão seja verdadeiro somente se os dois operandos sobre os quais se age forem de valor também verdadeiro. Caso o valor de um dos operandos seja falso, o resultado será falso. Existe, porém, uma diferença entre o **AND** do Apple e o do TRS-80 ou ZX81, quando esta instrução é utilizada numa expressão numérica. Suponha a operação:

```
C = A AND B
```

No Apple, o resultado em **C** será igual a **1** se **A** e **B** forem diferentes de zero, e igual a zero se pelo menos um dos dois for igual a zero. No TRS-80 e ZX81, o resultado em **C** será igual ao menor dos dois operandos (em termos absolutos) se ambos forem diferentes de zero, e zero se um dos dois, ou ambos, forem iguais a zero.

• **ASC ("X")** – Igual a **CODE ("X")** no ZX81. No TRS-80 e Apple, esta função retorna o valor decimal equivalente ao código ASCII utilizado para representar o carácter **"X"** ou o carácter mais à esquerda da cadeia de caracteres **"X".** No ZX81, o valor retornado não obedece ao padrão ASCII, e sim ao padrão interno da máquina.

• **ATN (X)** – Esta função, semelhante nos três equipamentos, nos retorna o arcotangente de um número ou expressão **X,** expresso em radianos.

• **BREAK** – No Apple, equivale ao acionamento conjunto das teclas **CONTROL** e **C.** Interrompe a execução de um programa.

• **CHR$ (X)** – No TRS-80 e no Apple, esta função nos retorna o caráter equivalente ao número ou expressão **X,** utilizado o código ASCII. No ZX81, contudo, o caráter retornado equivale ao padrão interno do equipamento, e não ao padrão ASCII.

• **CLEAR** – Também **CLR,** quando usado o "Integer BASIC" do Apple. Coloca todas as variáveis alfanuméricas com conteúdo nulo (não é o mesmo que espaço), e zera todas as variáveis numéricas. Após um **CLEAR,** as matrizes devem ser redimensionadas. No caso do TRS-80, este comando pode ter um argumento numérico (número ⩾ 0), o qual define (em bytes) o tamanho da área de memória a ser reservada para variáveis alfanuméricas.

• **CLOAD "X"** – No Apple, **LOAD,** e no ZX81, **LOAD "X".** Este comando carrega um arquivo na área de programa do equipamento. No TRS-80, o nome de ARQUIVO **"X"** é opcional. Se omitido, tal como acontece no Apple, será carregado o primeiro arquivo encontrado na fita. Tanto no TRS-80 como no ZX81, será utilizada apenas a primeira letra do nome fornecido.

• **CLS** – No Apple, equivale a **HOME.** Tem por finalidade apagar a tela e posicionar o cursor no canto superior esquerdo da mesma (exceção feita ao ZX81, no qual o cursor vai para o canto inferior esquerdo da tela).

• **CONT** – Ou também **CON,** no "Integer BASIC" do Apple. Continua a execução de um programa BASIC após a ocorrência de uma instrução **STOP** ou um **BREAK.**

• **COS (X)** – Esta função nos retorna o coseno de um número ou expressão **X,** expresso em radianos.

• **CSAVE "X"** – No Apple, **SAVE,** e no ZX81, **SAVE "X".** Salva o conteúdo da área de programa numa fita cassete. **"X"** é opcional para o TRS-80, obrigatório para o ZX81 e inexistente para o Apple. Somente o primeiro caráter do nome será utilizado para identificar o arquivo.

• **DIM** – Esta declaração é utilizada para a definição de matrizes de variáveis, tanto do tipo numérico quanto do tipo alfanumérico. No TRS-80 e no Apple, se uma variável não houver sido definida por uma declaração **DIM,** mas estiver sendo utilizada de forma subscrita, é assumido que cada um dos índices poderá variar de 0 a 10. No ZX81 é obrigatória a utilização de **DIM** para variáveis do tipo numérico. Ainda neste último, o índice varia a partir de **1** (e não de 0) até o valor máximo indicado na declaração **DIM.**

• **E** – Nos três equipamentos, a letra **"E"** é utilizada na representação de números em notação científica.

• **EXP (X)** – Esta função exponencia uma variável a uma potência indicada pelo número ou expressão **X.**

• **FOR...TO...STEP...:NEXT** – Nos três micros estes quatro elementos são utilizados para a confecção de loops controlados.

• **GOSUB N** – Nos três equipamentos, esta instrução permite executar uma sub-rotina que começa na linha **N.** Há, porém, que se fazer uma observação: no TRS-80 e no Apple, **N** tem que ser uma constante, enquanto que no ZX81 poderá ser também uma variável ou uma expressão.

• **GOTO N** – Esta instrução, comum a todas as três máquinas, executa um desvio incondicional para a linha número **N.** Quanto ao número **N,** cabe a mesma observação feita na instrução **GOSUB,** anteriormente descrita.

• **IF...THEN...** – Estes dois elementos permitem avaliar a veracidade, ou não, de uma expressão ou variável. Se for constatado que a condição é verdadeira, é executada a instrução que se segue ao **THEN.** Caso contrário, o computador executa a instrução da próxima linha.

• **INKEY$** – Esta função não existe no Apple, mas pode ser simulada, como veremos. Quando encontrada, tal função faz uma leitura do teclado. Se ele hou- »

ver sido pressionado, a função retorna o caráter ativado; caso contrário, é retornada uma cadeia com NULO. No Apple, pode-se utilizar:

```
IF PEEK (- 16384) > 127 THEN GOTO nnn
```

Se o conteúdo deste endereço for maior do que 127, isto indica que a tecla foi pressionada. Neste caso, o valor encontrado neste endereço será o valor ASCII da tecla mais 128, isto é:

```
Conteúdo de - 16384 = ASC ("tecla pressionada") + 128
```

• **INPUT** — Esta instrução está disponível nos três equipamentos para a aquisição de dados a serem introduzidos em uma ou mais variáveis, numéricas ou alfanuméricas, via teclado.

• **INT (N)** — Esta função nos retorna o maior número inteiro igual ou menor que o número ou expressão **N.** Adicionalmente, o TRS-80 nos fornece duas funções, que são: **CINT (N)** e **FIX (N).** A primeira faz a mesma coisa que o **INT (N),** porém com a ressalva de que **N** deve estar entre — 32768 e + 32767. Esta função torna-se, assim, mais rápida. O **FIX (N)** será igual a **INT (N)** se **N** for positivo. Se **N** for negativo, **FIX (N)** será igual a **INT (N) + 1.**

• **LEN ("X")** — Esta função nos retorna o número de caracteres contidos na cadeia **"X".**

• **LET** — Esta é a instrução de atribuição de valor a uma variável. No ZX81, o seu uso é obrigatório em declarações de atribuição.

• **LIST** — Lista todo o programa na tela.

• **LIST n** — Lista o programa da linha **n** até a sua última declaração, no ZX81. No TRS-80 e Apple, vamos encontrar **LIST x-y** (ou, ainda, **LIST x, y,** no Apple) e **LIST-X** (ou, ainda, **LIST, X,** no Apple). No primeiro caso, são listadas as linhas que vão do número **x** ao número **y.** No segundo, o programa é listado a partir da sua primeira linha até à linha **X.** No TRS-80 e Apple, **LIST n** causará o aparecimento apenas da linha especificada.

• **NEW** — O programa contido na memória é totalmente apagado.

• **NOT** — É um operador lógico que inverte o valor de uma variável ou expressão lógica.

• **OR** — Este é um operador lógico que fará com que o resultado de uma expressão lógica seja verdadeira se pelo menos um dos itens operados for verdadeiro. Para que o resultado seja falso, todos os operandos têm que ser falsos. Existe, porém, uma diferença na forma de operar do Apple para o TRS-80 e o ZX81, quando os operandos são números. Por exemplo, na expressão:

```
C = A OR B
```

No Apple, o resultado de tal operação fará com que **C** assuma o valor 0 se ambos, **A** e **B,** forem zero, e 1 se pelo menos um dos operandos for diferente de zero. Já no TRS-80 e no ZX81, o resultado em **C** será igual ao do maior dos dois operandos em questão (em termos absolutos). O valor de **C** só será 0 se ambos, **A** e **B,** forem iguais a zero.

• **PEEK (N)** — Obtém o valor inteiro contido na posição de memória de endereço **N.**

• **POKE (n,X)** — Coloca no endereço de memória indicado por **n** o número inteiro ou conteúdo da variável inteira **X.**

• **PRINT** — Mostra na tela o conteúdo de variáveis, expressões e constantes. No TRS-80 e ZX81, pode-se utilizar a função **TAB (N),** juntamente com o **PRINT** para fazer a tabulação. No Apple, porém, a tabulação deve ser feita com a utilização da instrução **HTAB (N).** No TRS-80, **N** pode variar de 0 a 255, utilizando-se módulo 128 no caso de **N** ser > 127. No ZX81, este valor pode ser qualquer um, mas será reduzido a módulo 32. No **HTAB, N** pode assumir valores de 1 a 40. A instrução **PRINT@ n** do TRS-80 vai encontrar uma boa simulação através do **PRINT AT x, y** do ZX81 e nas duas instruções **HTAB** e **VTAB,** do Apple. Como existe discrepância nas dimensões de tela destes três equipamentos, não posso especificar uma fórmula "milagrosa" para adaptações, mas aqui vão algumas dicas para o amigo aí resolver o seu caso. Primeiro, as dimensões de cada um:

```
TRS-80: 16 linhas x 64 colunas (ou 32 colunas)
Apple : 24 linhas x 40 colunas
ZX81  : 22 linhas x 32 colunas
```

Agora vamos às instruções. O **PRINT @ n** do TRS-80 significa que a impressão será feita na posição absoluta número **n** da tela. E o que vem a ser isso? É uma forma de se visualizar a tela não como uma matriz de 16 linhas por 64 colunas, e sim como uma matriz unidimensional de 1024 posições, com estas posições variando de 0 a 1023. Assim, um elemento, ao invés de ser endereçado como posicionado, por exemplo, na linha 2 coluna 4, seria endereçado como posicionado no ponto 67 (64 posições da linha 1 + 4 colunas da linha 2, menos 1).

O **PRINT AT x, y** do ZX81 faz exatamente o inverso. Ele endereça as posições da tela como se esta fosse uma matriz bi-dimensional. Ou seja, indica que a impressão deve ser feita na linha **x,** coluna **y.** O Apple dispõe das instruções **HTAB (X)** e **VTAB (X),** que permitem o posicionamento do cursor em qualquer ponto da tela, cuidando a primeira do posicionamento na coluna e a segunda do posicionamento na linha.

• **REM** — Esta instrução serve para especificar uma linha de comentários.

• **RESET (x, y)** — No Apple **PLOT (x, y)** e no Sinclair **UNPLOT (x, y).** Esta instrução apaga um ponto gráfico anteriormente plotado na linha x coluna y. No Apple, o **PLOT** serve tanto para acender um ponto apagado, como para apagar um ponto anteriormente aceso. Nos outros dois equipamentos existem instruções específicas tanto para apagar como para acender um ponto (vide **SET (x, y).**

• **RETURN** — Provoca o retorno de uma sub-rotina para a instrução seguinte ao último **GOSUB** executado.

• **RND** — Gera randomicamente um número entre 0 e 1 (exclusive 1). No TRS-80, a instrução **RANDOM** assegura que o número gerado ao se executar a função **RND** seja realmente aleatório. No ZX81, tal é assegurado através da instrução **RAND.**

• **RUN n** — Roda um programa a partir da linha **n.**

• **SET (x, y)** — No Apple e ZX81: **PLOT (x, y).** Esta instrução é utilizada para ligar um ponto gráfico na tela, localizado na linha x, coluna y. No Apple, a instrução **PLOT (x, y)** também serve para apagar um ponto anteriormente aceso (vide instrução **RESET).**

• **SGN (N)** — Dirá se **N** é um número ou expressão positivo, negativo ou igual a zero. Para isso, retornará, respectivamente, os valores 1, —1 e 0.

• **SIN (N)** — Esta função retorna o seno de um número ou expressão **N** fornecido em radianos.

• **SQR (N)** — Esta função retorna a raiz quadrada do número ou expressão **N.**

• **STOP** — Esta instrução pára a execução do programa, retornando o controle ao usuário.

• **STR$ (N)** — Retorna o valor do número ou expressão **N** sob a forma de uma cadeia de caracteres numéricos.

• **TAN (N)** — Esta função retorna a tangente do número ou expressão **N** fornecido em radianos.

• **USR (n)** — Esta função merece uma atenção especial, em se tratando do TRS-80 e do Apple. No ZX81, **n** é o endereço de uma rotina em linguagem de máquina previamente colocada na memória através de uma instrução **REM** ou **POKEs** sucessivos. Neste equipamento, ao se voltar da rotina, a função **USR** nos fornece o conteúdo do par de registradores **BC.** No Apple, o valor **n** pode conter um parâmetro numérico, o qual será colocado nas posições que vão de **9D** (HEX) até **A3** (HEX). Antes do **USR** ser executado, nas posições que vão de **0A** a **0C,** deve ser colocada uma instrução **JMP HHHH,** onde **HHHH** é o endereço da sub-rotina de Assembler. Ao se retornar da função **USR,** esta fornecerá o conteúdo do registrador **A.** Se você não necessita que sejam passados parâmetros para a sua rotina em Assembler, o Apple dispõe da instrução **CALL n,** que transfere imediatamente o controle para a rotina encontrada no endereço **n.** No TRS-80, a instrução **USR (n)** funciona de forma parecida ao Apple. O valor **n** (deverá estar entre — 32768 e + 32767) é um parâmetro que poderá ser passado para a rotina de Assembler. O endereço para o qual se deseja que o **USR** transfira o controle tem que ser previamente colocado nas posições 16526 (LSB) e 16527 (MSB) através de instruções **POKE.** Para se obter o parâmetro que se deseja passar para a rotina em Assembler, é necessário ter como primeira instrução dessa rotina um **CALL 0A7FH.** Após a execução deste **CALL,** o valor dado através de **n** estará no par de registradores **HL.** Para passarmos um valor de volta da rotina chamada, devemos colocá-lo no par de registradores **HL** e encerrar a rotina com uma instrução **JP 0A9AH,** ao invés da instrução **RET.**

• **VAL ("X")** — Esta função vai tentar avaliar a cadeia de caracteres como se fosse um número ou uma expressão numérica, retornando o seu valor numérico equivalente.

No próximo número, continuaremos com o nosso dicionário. Até lá.

Orson Voerckel Galvão é Analista de Sistemas da Petrobrás Distribuidora S.A., no Rio de Janeiro, e Assessor Técnico de MICRO SISTEMAS. Orson foi o autor do Curso de BASIC publicado nos números 2 a 9 de MICRO SISTEMAS.

Pascal

Criada em 1971, Pascal já é a segunda linguagem mais usada nos micros dos E.U.A. Veja o porquê deste sucesso.

No ensino ou na ciência, um poderoso instrumento

Mauricio Costa Reis

Para melhor entender o que é Pascal e o porquê de sua crescente utilização nos micros computadores, é preciso recordar o seu surgimento. Niklaus Wirth, professor em Zurique, era um apaixonado pelo ensino da computação; para ele a linguagem de programação deveria refletir os conceitos mais importantes e fundamentais da computação, de forma clara e natural. Desta forma, o programa resultante poderia ser entendido por qualquer pessoa, sem dificuldades.

Dos estudos de Wirth (1) surgiu em 1971 a linguagem Pascal — nome dado em homenagem ao conhecido matemático Blaise Pascal do século XVII que, entre outras invenções, criou a máquina de calcular. Uma das causas da grande disseminação da linguagem Pascal, que a torna fácil de aprender e adequada aos iniciantes, além de poderosa para as grandes aplicações, é a possibilidade de se poder praticar a Programação Estruturada (PE).

O que é PE? A PE introduz conceitos (2) que possibilitam "dividir" o programa em vários "pedaços" bem conhecidos que, quando "agrupados" irão compor o programa. Diz-se então que temos um "programa estruturado". A vantagem desta divisão é que fica mais fácil e rápido construir um programa correto.

Existem outras linguagens que permitem que se pratique a PE com muita facilidade, como o ALGOL. Já o BASIC, FORTRAN e o COBOL são conhecidas como linguagens "não-estruturadas", porque dificultam a estruturação.

ALGUMAS PARTICULARIDADES

Podemos mencionar algumas peculiaridades do Pascal, que o diferencia principalmente do BASIC (3):

- **Comandos Compostos** — é o agrupamento de vários comandos simples para simbolizar uma idéia mais complexa. Os comandos compostos começam por **BEGIN** e terminam por **END**;

- **Declaração de variáveis** — deve-se declarar todas as variáveis usadas no programa. Desta forma todas as variáveis e informações sobre estas (tipo, significado etc.) ficam documentadas;

- **Tipos especiais de variável — SET:** semelhante à definição de "conjunto" utilizada na Matemática, permite as mesmas operações (união, intersecção, pertence etc.); **POINTER:** facilita o uso de estrutura de dados, tais como filas e pilhas de dados; **RECORD:** assemelha-se à estrutura de dados do COBOL;

- **Definição do tipo de variável** — além dos tipos comuns (**REAL, INTEGER, LOGICAL** etc.), a linguagem Pascal permite a criação de tipo de variável pelo programador.

Neste último caso, de definição de tipo de variável, pode-se ter, por exemplo, o seguinte: **TYPE** ESTCIVIL = (SOLTEIRO, CASADO, VIÚVO, DESQUITADO); **VAR** PESSOA = ESTCIVIL. Com isto, estamos criando o tipo ESTCIVIL e afirmando que a variável PESSOA é deste tipo. Embora aparentemente complicado, este é um dos detalhes que destacam o Pascal das outras linguagens, e na prática é muito fácil de entender e usar.

A entrada e saída (E/S) é feita utilizando-se os comandos **READ, READLN, WRITE, WRITELN, RESET, REWRITE, GET, PUT, EOF** e **EOLN.** Os mais usados, entretanto, são **READ** (entrada) e **WRITE** (saída), mas todos são muito simples de serem usados.

O Pascal possui ainda vetores, sub-rotinas (inclusive com recursividade), rotinas matemáticas (SIN, COS, ABS etc.) e rotinas de tratamento de cadeia de caracteres (Strings), além dos seguintes comandos de decisão e repetição:

```
IF <condicao> THEN <comando>;
IF <condicao> THEN <comando> ELSE <comando>;
CASE <exp.> OF
 <valor>: <comando>;
 . . .
 <valor>: <comando>;
 WHILE <condicao> DO <comando>;
REPEAT <comando> UNTIL <condicao>;
FOR <var.>:=<exp.> TO <exp.> DO <comando>;
```

Como se pode observar, o comando **CASE** do Pascal é semelhante a vários **IFs** do BASIC. E para facilitar o uso do Pascal em microcomputadores, esta linguagem foi acrescida com rotinas para simplificar o uso da tela, como, por exemplo, **GOTOXY** para posicionar o cursor na tela, e **READXY,** que fornece a posição atual do cursor.

VERSATILIDADE DE APLICAÇÃO

Com o crescimento da utilização dos microcomputadores, o Pascal vem sendo muito aplicado no ensino da computação, principalmente pela simplicidade e clareza dos programas. Mas não só no ensino, porque além de uma linguagem geral e acessível, é poderosa, com vários recursos para aplicações comerciais e científicas.

O Pascal pode ser compilado ou interpretado, mas a versão interpretada é a mais difundida porque exige menos espaço de memória e, por isto, é a mais encontrada nos micros. Nos E.U.A., por exemplo, existem vários micros que têm Pascal: o Atari e o micro pessoal da IBM, por exemplo. Há também uma versão do Pascal para o CP/M (Osborne, Apple-CP/M etc.) que pode ser colocada em qualquer micro com sistema operacional CP/M. Além disto, vários tipos de Pascal podem ser encomendados em qualquer software-house americana. (veja no quadro o endereço de algumas delas).

PASCAL X BASIC

A melhor forma de ilustrar a diferença entre a primeira linguagem mais usada nos micros dos E.U.A., o BASIC, e o Pascal, é observar o exemplo destacado na figura 1, que calcula o MDC (máximo divisor comum) entre dois números. Um exemplo simples, onde se pode observar algumas (apenas algumas) particularidades do Pascal. Como se pode constatar, de início, o Pascal, ao contrário do BASIC, tem formato livre. O comando **DIV** é usado para divisão inteira (sendo que não era necessário no programa Pascal) e **MOD** fornece o resto da divisão de N1 por N2. O comando composto, neste exemplo, está entre os comandos **REPEAT** e **UNTIL** (ao invés de **BEGIN** e **END**).

```
PROGRAM CALCMDC (INPUT, OUTPUT);
CONST ZERO = 0;
VAR MDC, N1, N2, D, R : INTEGER;
BEGIN
   WRITE ('N1 ?'); READ (N1);
   WRITE ('N2 ?'); READ (N2);
   REPEAT
      MDC := N2;
      D := N1 DIV N2;
      R := N1 MOD N2;
      N1 := N2;
      N2 := R;
   UNTIL R = ZERO;
   WRITE ('M.D.C.=',MDC);
END.
```

```
10 REM CALMDC
20 INPUT "N1 ";N1
30 INPUT "N2 ";N2
40 MDC=N2
50 D=INT(N1/N2)
60 R=N1-D*N2
70 N1=N2
80 N2=R
90 IF R<>0 THEN 40
100 PRINT "M.D.C.=";MDC
110 END
```

Figura 1 — O mesmo cálculo em duas versões: Pascal e BASIC.

Embora o programa em Pascal seja maior que o em BASIC, aquele está melhor documentado e muito mais fácil de entender, até mesmo por quem nunca o viu antes... Por estas e outras características, o Pascal tem tudo para se tornar a linguagem desta década, como foram o FORTRAN e o COBOL, porque reúne duas qualidades raramente encontradas juntas: é simples e poderoso. Não é por acaso, como se pode verificar, que o Pascal está sendo muito utilizado nos micros. O BASIC que se cuide!

BIBLIOGRAFIA

(1) WIRTH, Niklaus, **Programação Sistemática com Pascal,** Ed. Campus, 2ª edição, Rio de Janeiro, 1982.
(2) OLE, Johan Dahl, **Structured Programming,** London Ed. Academic Press (A.P.I.C. Studies in Data Processing), Londres, 1972.
(3) Esperança, Cláudio, **Pascal,** Apostila do Curso de Pascal do NCE/UFRJ, Rio de Janeiro, 1980.

Software-houses americanas

— *Softech Microsistems, Inc:* 9494 Black Mountain Rd, San Diego, CA 92126 (UCSD-Pascal, vendido para o micro pessoal da Texas Instruments);
— *JRT Systems:* 1891 23rd Avenue, San Francisco, CA 94122 (Pascal para CP/M);
— *Atari, Inc:* Dept CiZ P. D. Box 16525, Denver, CO 80126;
— *IBM Personal Computer:* External Submissions, Dept. 765 PC Armonk, New York, 10504;
— *Discount Software Group:* 6520 Selma Ave., Suite 309, Los Angeles, CA 90028 (Pascal/MT+, Pascal/M, Tiny-Pascal).

Mauricio Costa Reis é formado em Informática pela UFRJ, atualmente trabalha como Analista de Sistemas na Portobrás e é Consultor Técnico da loja Microcenter Informática Ltda., no Rio de Janeiro.

COBOL

Entre as linguagens de programação, COBOL é a mais indicada para tratamento de arquivos e estrutura de dados.

A primeira em aplicações comerciais

José Luiz do Nascimento Silva

Por definição, uma linguagem de alto nível é aquela cujo estilo e recursos de programação se aproximam da linguagem natural do programador. Caracteriza-se por sua independência às especificações de arquitetura de qualquer computador, ao conjunto interno de instruções da máquina, tamanho da palavra, velocidade e tipo de processador, dispositivos de entrada e saída etc. Esta independência implica em que as linguagens de alto nível sejam implementadas através de programas sofisticados e complexos como os compiladores, interpretadores, ligadores etc.

É fácil constatar que o leque de linguagens foi se abrindo proporcionalmente ao surgimento de novas necessidades, que para seu atendimento e solução precisavam de mais recursos, fossem eles em termos de instruções, velocidade de processamento e até mesmo compatibilização entre equipamentos. Desta forma, cada fabricante de computador dava o seu "toque pessoal" na implementação do seu software básico (sistema operacional, compiladores, ligadores, interpretadores etc), incompatibilizando os sistemas (softwares de aplicação) com os demais equipamentos.

Analisando a figura 1 notamos que existem inúmeros usuários com a mesma necessidade e que, portanto, utilizam-se de um mesmo sistema, porém em equipamentos diferentes. Ora, se não houver uma padronização nas instruções da linguagem de programação

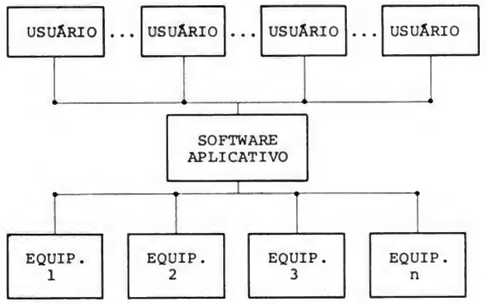


Figura 1: Relação entre usuários, software de aplicação e equipamentos

utilizada, teremos que repetir todo o trabalho de desenvolvimento, programação, testes e implementação ao transferirmos um mesmo programa para outro equipamento, o que se torna altamente oneroso ao usuário, além de atrasar a efetiva produção e conseqüente solução do problema.

Grandes esforços vêm sendo realizados em prol da padronização e regulamentação das linguagens de programação de modo a reduzir tais problemas. Podemos citar como exemplo, o ANSI (American National Standards Institute).

COBOL – COMMON BUSINESS ORIENTED LANGUAGE

Criada em 1959 por um comitê de grandes usuários, fabricantes e órgãos do governo americano, hoje em dia é, de longe, a linguagem mais utilizada no mundo inteiro em aplicações comerciais. Seus criadores buscavam uma linguagem que fosse comum, que permitisse a transferência de programas e técnicas de programação entre equipamentos diferentes. No entanto, apesar dos esforços do ANSI, o resultado não foi uma linguagem verdadeiramente comum a todos os computadores.

Sua orientação, como dissemos, é comercial, com extensas facilidades para manuseio de arquivos e estrutura de dados. COBOL não é utilizado em aplicações científicas mais pela ineficiência dos compiladores no tratamento deste tipo de informação do que propriamente pela deficiência da linguagem.

A estrutura do COBOL é semelhante a da língua inglesa, onde é limitado o número de palavras com sintaxe própria. Num programa COBOL existem quatro divisões que auxiliam na simplicidade, eficiência e leitura, cada uma apresentando finalidades básicas:

1 – **Identification Division:** identifica o programa-fonte (nome, autor, data) e o módulo-objeto obtido na compilação.

2 – **Environment Division:** especifica o equipamento mais indicado tanto para a compilação quanto para a execução, e assinala os arquivos de dados a serem utilizados no programa e seus respectivos dispositivos periféricos.

3 – **Data Division:** descreve os dados que o programa aceitará como entrada e os que serão produzidos como saída, assim como os dados e constantes intermediários do programa.

4 – **Procedure Division:** descreve todas as ações executadas pelo programa, como as operações e manipulações a serem executadas com os dados, controles etc.

À excessão da **Identification Division,** as demais divisões poderão conter seções que, segundo a estruturação do COBOL, identificarão um determinado propósito, sendo usadas sempre que necessárias. As mais utilizadas são:

A) **Environment Division**

-- **Configuration Section**

Nesta seção teremos a informação sobre o computador com o qual iremos trabalhar e os mnemônicos a serem aplicados para associar funções especiais, como salto de página na impressora, marcação da vírgula como ponto etc.

– **Input-Output Section**

Teremos aí a associação do arquivo de dados físico e seu respectivo dispositivo de armazenamento ao nome do programa, bem como a descrição do tipo de organização do arquivo, o método e a chave de acesso.

B) **Data Division**

-- **File Section**

Nesta seção serão descritos os dados, ou estrutura dos registros, referentes aos arquivos assinalados na **Input-Output Section.**

– **Working-Storage Section**

Nesta seção serão descritas todas as variáveis de trabalho pelo programa.

– **Linkage Section**

Onde serão descritos e definidos os dados passados ou recebidos, comuns a outros programas.

C) **Procedure Division**

Nesta divisão não há nenhuma seção em especial, exceto em duas situações:

1 – Quando empregamos o comando

```
SORT .... INPUT PROCEDURE A
          OUTPUT PROCEDURE B
```

, onde **A** e **B** devem ser obrigatoriamente um nome de seção. Exemplo:

```
A= INICIO-SORT SECTION
B= SAIDA-SORT SECTION
```

2 – Em micros, no caso de programas segmentados, deve-se colocar seções com número superior a 55. Por exemplo:**(OVR1–ROT SECTION 56).**

Logo após as divisões e seções, num penúltimo nível, estão os parágrafos. Estes, quando pertencentes às três primeiras divisões, possuem nomes pré-definidos e, quando pertencentes à **Procedure Division,** assumirão o nome definido pelo próprio programador.

A função de um parágrafo é documentar, identificar um tipo de ação, mas quando na **Procedure Division,** além de documentar, atua como referência para desviar uma execução (comando **GO TO** nome-de-parágrafo). Todos os nomes relativos a seções e parágrafos podem conter até 32 caracteres.

Num último nível, aí então vêm as instruções – verbos que comandam uma ação. Hierarquicamente, portanto, COBOL é formado por: divisões, seções, parágrafos e instruções.

ARQUIVOS

O COBOL possui as seguintes características quanto à manipulação de arquivos:

Acesso – Sequencial
– Direto

Organização – Sequencial
– Sequencial-indexada
– Relativa

Com relação às organizações de arquivo que permitem acesso direto, e por consequência sequencial, faremos algumas considerações a respeito da **chave de busca.**

● **Organização Indexada**

A chave ***pode ser*** numérica ou alfanumérica, cujo tamanho não deverá exceder a 15 bytes, e onde somente a área de índices (chaves) está em ordem crescente. A área de dados está na ordem da gravação dos registros e contém a chave de acesso.

Considerando um arquivo de organização sequencial-indexada, cujo lay-out está representado na figura 2, veja, após a gravação de dados, como ficam as áreas de índices e dados. Este esquema tem efeito apenas de esclarecimento ao leitor, pois os controles e as estruturas de endereçamento são bem mais complexas.

Observação: os registros na área de dados são gravados na ordem de lançamento; a área de índices está ordenada alfabeticamente.

»

• Organização Relativa

A chave **deve ser** numérica e o registro (dado) ocupará a posição física no arquivo, idêntica ao valor da chave (posição relativa). Não importa, por exemplo, a existência ou não de outros registros no arquivo, mas se gravarmos um dado cujo valor da chave a ele atribuído seja 8, este dado estará no registro número 8 e será o oitavo na ordem de registros antecedentes.

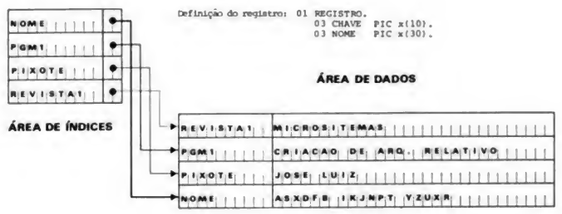


Figura 2: Exemplo de arquivo de organização sequencial-indexada

Figura 3: Exemplo de arquivo de organização relativa

Suponhamos agora que transformamos o arquivo da figura 2 em relativo, reservando para ele uma área de dados para vinte registros, gravando, respectivamente, as chaves 1, 3, 4 e 20. Na figura 3 podemos ver como ficará o arquivo, onde a área de gravação, neste caso, não importa. Os registros ocuparão a posição relativa do valor da chave na área reservada para o arquivo.

Neste tipo de organização, os dados sempre estarão ordenados por chave, que é a própria posição física ocupada no arquivo, se bem que esta não faz, necessariamente, parte do registro. O primeiro registro será sempre o de chave igual a **1.** Em micros, a única forma para se reservar área para os registros é, no ato da criação do arquivo, gravar o registro de chave igual a **n+1**, onde **n** é o número máximo de registros que o arquivo deverá conter.

VARIÁVEIS

COBOL utiliza variáveis numéricas, alfanuméricas, alfabéticas e de edição (existem alguns outros tipos que podem ser atribuídos, mas são pouco usados). Uma variável pode conter até 32 caracteres normais (não especiais).

O tamanho máximo de representação de dados numéricos é de 15 dígitos. Sua representação interna pode ser reduzida em número de bytes utilizados, definindo-se a variável como sendo compactada **(COMP, COMP-3, COMP-4).**

Como o COBOL não trabalha com a representação de dados referentes às variáveis numéricas em ponto flutuante, precisamos ter certeza da **PICTURE** definida (máscara de representação interna do número), para não haver perda de números. Considere, por exemplo, a variável **WS-RESULT,** estabelecida como **PIC 9(9)V9(4).** Os números que poderão ser representados por ela estão no intervalo de – 99.999.999,9999 a 999.999.999,9999. Podemos ver que a variação se dá em termos do número (sempre fixo) de casas inteiras e decimais, e não em relação ao número de bytes total da variável, que são 13.

Em COBOL temos que definir exatamente o que deve representar a variável. A ela estará preso o resultado; caso contrário, se for superada a margem de representação, estaremos sujeitos à ocorrência de erros.

As variáveis alfabéticas, como o nome diz, são aquelas que só podem representar letras, atribuindo-se ao nome da variável uma **PICTURE** que defina sua representação que, neste caso, é "A": **WS–ALFA PIC AA.**

As variáveis de edição são aquelas que servem para exibir os resultados numa forma editada, eliminando-se os zeros à esquerda, colocando-se pontos nas divisões de classe dos números, editando a vírgula como ponto decimal etc. »

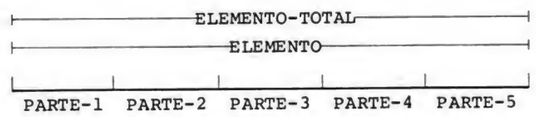
PICTURE	NÚMERO A SER EXIBIDO	EXIBIÇÃO
9(9)V99	1458319	00145831900
zzz.zzz.zz9,99	1458319	1.458.319,00

Existem outros tipos de edição que podem ser conjugados, tais como: asteriscos, cifrão, sinal (soma e subtração) — colocados à esquerda do número — e alguns outros mais.

O COBOL não possui funções ou instruções que manipulem strings dentro de variáveis. Para tal seria necessário desmembrá-las dígito a dígito. Por exemplo: consideremos a seguinte estrutura de dados:

```
01 ELEMENTO-TOTAL
   03  ELEMENTO     PIC X(5)
   03  ELEMENTO-R REDEFINES ELEMENTO.
       05 PARTE-1    PIC X.
       05 PARTE-2    PIC X.
       05 PARTE-3    PIC X.
       05 PARTE-4    PIC X.
       05 PARTE-5    PIC X.
```

No programa, se utilizarmos a variável **ELEMENTO-TOTAL** ou **ELEMENTO,** o efeito será o mesmo, não alterando o resultado final. Mas se aplicarmos qualquer uma das variáveis apresentadas como nível **05** neste exemplo anterior, dependendo da variável, estaremos usando um caráter dentro da variável total. Desta forma, temos a seguinte representação x associação:



Assim como utilizamos este artifício para "dividir" um string, poderíamos utilizá-lo para "concatenar" as pequenas partes para formar o "todo". Dependendo da aplicação, isto pode ser desastroso devido ao acréscimo sensível do número de linhas a serem codificadas para a realização destas operações, mas em geral a estruturação das variáveis no COBOL permite um bom e eficiente resultado dada a orientação desta linguagem, além do alto efeito na documentação e depuração do programa.

COBOL EM MICROS

Seguindo a mesma linha dos grandes equipamentos, cada fabricante de micros costuma implementar seu compilador de modo a obter e dispor do maior número possível de instruções e recursos. O resultado é a grande diferença entre os **SETS** de instruções de um para outro micro. Contudo, uma das funções que apresentam as maiores diferenças quanto aos compiladores é a que se refere ao tratamento de telas. Por serem máquinas orientadas para o processamento interativo, os micros utilizam mais esta função do que os grandes sistemas "batch".

Para o tratamento através de um programa COBOL não existe uma padronização, pois cada fabricante desenvolve e implementa seu software conforme as características de seu equipamento, objetivando obter os melhores efeitos e resultados. Podemos citar alguns exemplos:

- criação de uma seção na **Data Division** para definição da tela: **Screen Section/Terminal Section;**
- extensões ao comando **DISPLAY** e **ACCEPT** para trabalhar com tabulação no vídeo e associação automática;
- chamadas (**CALL**) de rotinas do sistema operacional para manipulação de tela (funções no vídeo).

Os usuários de COBOL em microcomputadores devem se preocupar principalmente com o tamanho do programa. Quando da compilação e posterior link-edição, poderá ocorrer erro por falta de espaço na memória. Para solucionar casos como este, os micros, em sua maioria, segmentam o programa, formando uma estrutura de **OVERLAY**. Para tal, basta verificar no programa as rotinas que sofrem menos chamadas e colocá-las como seções, atribuindo-se a elas um número superior a 55, como por exemplo: **CRITICA-DATA SECTION 55.** Desta forma, o compilador vai gerar o código relativo à seção independente do corpo principal do programa, trazendo-o para a memória somente quando for necessária. Não se deve abusar muito desta técnica, pois prejudicará a execução do programa se houver muitas "cargas".

COMPILADOR E INTERPRETADOR

O COBOL interpretado é basicamente composto de um compilador que gera um pseudo código-objeto e um "Run-Time System" que interpreta este código durante a execução, após carregá-lo na memória. O interpretador inicia a execução com o primeiro comando da **Procedure Division,** e segue sequencialmente, a menos que a operação indique um desvio.

Toda vez que fazemos referência a uma variável ou outro objeto, o interpretador necessita acessar suas tabelas de descrição e obter as informações necessárias. As pseudo-operações contêm o código da operação a ser efetuada e o endereço das entradas nas tabelas referentes aos operadores especificados. De maneira geral o código-objeto é compacto e facilita a fusão de rotinas externas (reentrância).

A versão interpretada apresenta, por outro lado, um aspecto negativo: o alto "overhead" inerente à própria interpretação, muitas vezes superior a cinco ou seis vezes à execução da mesma função por código diretamente executável.

As versões compiladas geram um código-objeto que, após a resolução dos endereços externos pelos ligadores, é executado pelo próprio firmware do computador. Os programas-objeto gerados por estas versões tendem a ser maiores do que os interpretadores e muitas vezes não implementam reentrância. Apesar da maioria das versões compiladas não se preocupar em gerar um código otimizado, elas são mais eficientes do que as versões interpretadas.

PARALELO COM BASIC

Como já dissemos, COBOL é uma linguagem de programação voltada para a área comercial. Mas no caso de micros — principalmente os que possuem as versões interpretada e compilada do BASIC — em termos de aplicação, perde o sentido dizer que o COBOL sobrepõe-se ao BASIC nesta área. Isto porque, embo-

COBOL	BASIC
CLOSE	CLOSE
A OCCURS 10 TIMES	DIM A(10)
VARYING I FROM 1 BY 1 UNTIL I > 10	DO I=1 TO 10
PERFORM	GOSUB
———	NEXT I
GO TO	GO TO
———	RETURN
IF J=8	IF J=8
IF ALFA = 'ABC'	IF A$ = "ABC"
ACCEPT C	INPUT C
READ nome de arquivo	INPUT # 3,R$
OPEN	OPEN
DISPLAY I J K	PRINT I;J;K
WRITE nome de dado	PRINT # 3;R$
* (na coluna 7) comentário	REM
STOP RUN	STOP ou END
COMPUTE K= X+3,14*R	K= X+3.14*R

Figura 4: Paralelo entre comandos COBOL e BASIC

COBOL	BASIC
Requer um tempo razoável para seu aprendizado.	Requer pouco tempo para o aprendizado.
Possui maior número de instruções.	Possui menor número de instruções.
Possui menor número de funções.	Possui maior número de funções.
Usada essencialmete em aplicações comerciais.	Usada tanto em aplicações científicas como comerciais.
Alto grau de estruturação.	Grau de estruturação razoável.
Facilidade no entendimento de programas (acompanhamento da lógica).	Entendimento de programas (acompanhamento da lógica) mais trabalhoso.
Requer mais tempo na construção de um programa (alta verbosidade).	Requer pouco tempo na construção de um programa (baixa verbosidade).
Grande efeito auto-documentativo (nomes de variáveis, rotinas, etc...).	Baixo efeito auto-documentativo.
Alta eficiência na utilização de arquivo de dados	Requer elaboração de artifícios e algoritmos para utilização de arquivos de dados (máquinas que possuem o BASIC standard - arquivo sequencial e randômico).
Existência de poucos dialetos (maior compatibilidade entre os diversos equipamentos).	Existência de vários dialetos (menor compatibilidade entre os diversos equipamentos).
Programas extensos, portanto ocupam mais área em disco e memória.	Programas menores, portanto ocupam menor área em disco e memória.
Variáveis têm tamanho e formato fixo no programa.	O tamanho e o formato das variáveis devem ser controlados.

Figura 5: Comparação COBOL x BASIC

ra tenha sido criado para atender a propósitos científicos e facilitar o manuseio para o usuário-programador, o BASIC também atende, com razoável eficiência às atividades da área comercial.

Já existem sistemas de contabilidade, folha de pagamento, controle de estoque, contas a pagar/receber, mala direta, compras etc, todos desenvolvidos em BASIC. É claro que, quando se fala em tratamento de arquivos, existe um "break" no BASIC. No entanto, podemos nos valer de artifícios e algoritmos para superar estes problemas. Não há dúvida que se estes sistemas fossem desenvolvidos em COBOL seriam mais eficientes, já que as características desses sistemas se adequam mais aos recursos desta linguagem.

Na figura 4 podemos ter uma idéia de alguns comandos do COBOL e do BASIC que possuem correlação. Comparar as duas linguagens na hora de desenvolver um sistema é uma questão de, primeiramente, avaliar as necessidades do usuário, e em segunda ver qual das duas se adequará com mais eficiência (ver figura 5).

De acordo com o que os fabricantes divulgaram no XV Congresso Nacional de Informática e II Feira Internacional de Informática, os equipamentos que possuem como linguagem de programação o COBOL e o BASIC são: Brascom BR100, Cobra C305, Labo 8221, Logus LOG Z80, Polymax 201DP, Prológica Sistema 700, Quartzil QI800, Microscopus, SID 3000, Sisco MB 8000SM, Edisa ED281 e Schumec M100/85.

Para dar ao leitor uma visão de um programa COBOL e estabelecer um paralelo com um programa BASIC, apresentamos uma listagem em cada linguagem. A função de ambas é gerar um arquivo (relativo no COBOL e randômico no BASIC), contendo simplesmente um nome que é informado pelo teclado e sua respectiva chave de acesso.

Nesta comparação, observe também o tratamento de telas de um e outro programa (note que o programa COBOL utiliza a **Screen Section,** não comum a todos os equipamentos, existente apenas no SID 3000, Prológica Sistema 700 e Schumec).

Programa COBOL para geração de arquivo relativo

```
IDENTIFICATION DIVISION.

PROGRAM-ID.    GERARQ.
AUTHOR         JOSE LUIZ N SILVA.
DATE-WRITTEN. 22/02/83.
*REMARKS.
* ESTE PROGRAMA GERA UM ARQUIVO RELATIVO
* CONTENDO NOMES QUE SERAO FORNECIDOS VIA
* TECLADO, E A CHAVE DE ACESSO ESTABELE-
* CIDA PELO USUARIO.

* ASSINALACAO DO EQUIPAMENTO E ARQUIVOS
 ENVIRONMENT DIVISION.

 CONFIGURATION SECTION.

 SOURCE-COMPUTER.  SID-3000.


 INPUT-OUTPUT SECTION.

     SELECT  CADASTRO
       ASSIGN TO  DISK
       ORGANIZATION IS RELATIVE
       ACCESS MODE IS RANDOM
       RELATIVE KEY IS WS-RELATIVE.

* DEFINICAO DOS REGISTROS,VARIAVEIS E TELAS
 DATA DIVISION.

 FILE SECTION.

 FD  CADASTRO
     LABEL RECORD IS STANDARD
     VALUE OF FILE-ID 'A:CADASTRO'.

 01  REGISTRO.
     03  NOME            PIC X(30).


 WORKING-STORAGE SECTION.

 77  WS-RELATIVE       PIC 999  COMP.
 77  WS-NADA  PIC X.
 77  WS-CHAVE PIC 999.
 77  WS-NOME  PIC X(30).
 77  WS-PONTO PIC X(30)  VALUE ALL '.'.


 SCREEN SECTION.


 01  TD01-TELA-NOMES.
     03  LINE 7  COLUMN 20
         VALUE 'CHAVE :,...'.
     03  LINE 9  COLUMN 20
         VALUE 'NOME  :'.
     03  LINE 9  COLUMN 28
         PIC X(30) FROM WS-PONTO.

* ACOES A SEREM EXECUTADAS
 PROCEDURE DIVISION.

* LIMPAR A TELA, ABRIR ARQUIVO

     DISPLAY (1, 1) ERASE.


     OPEN I-O CADASTRO

* LOOPING DE DIGITACAO E GRAVACAO

 LOOP-LE-GRAVA.

     DISPLAY (1, 1) ERASE.
     DISPLAY TD01-TELA-NOME.
     ACCEPT (7, 28) WS-CHAVE.
       IF  WS-CHAVE = ZEROS  GO TO FIM.
* TESTA VALIDADE DA CHAVE

     MOVE WS-CHAVE TO WS-RELATIVE.
     READ CADASTRO INVALID KEY
                   GO TO RECEBE-NOME.

 JA-EXISTE.

     DISPLAY (23, 1)
      'JA EXISTE REGISTRO COM ESTA CHAVE '
     ACCEPT WS-NADA.
     GO TO LOOP-LE-GRAVA.

 RECEBE-NOME.

     ACCEPT (9, 28) WS-NOME.

 PREPARA-REG-GRAVA.

     MOVE WS-CHAVE TO WS-RELATIVE.
     MOVE WS-NOME  TO NOME.
     WRITE REGISTRO INVALID KEY
      DISPLAY (23, 1)
       'ERRO NA GRAVACAO '
      ACCEPT WS-NADA.
     GO TO LOOP-LE-GRAVA.

 FIM.

     DISPLAY (1, 1) ERASE.
     CLOSE CADASTRO.
     STOP RUN.
```

Programa BASIC para geração de arquivo randômico

```
10  REM       ESTE PROGRAMA GERA UM ARQUIVO
20  REM       RANDOMICO CONTENDO NOMES QUE SE-
30  REM       RAO FORNECIDOS VIA TECLADO, E A
40  REM       CHAVE DE CCESSO ESTABELECIDA PE-
50  REM       LO USUARIO
60  REM       * JOSE LUIZ N SILVA - 22/02/83 *
70  REM       PROGRAMA PARA EQUIPAMENTOS COM
80  REM       BASIC DA MICROSOFT
90  REM
100 REM       DEFINICAO DA FUNCAO DE TABULACAC
110 REM       NO VIDEO
120 DEF FNT$(L%,C%)=CHR$(27)+"Y"+CHR$(32+L%)+
    CHR$(32+C%)
130 REM       LIMPAR A TELA E ABRIR ARQUIVO
140 PRINT CHR$(12)
150 OPEN "R",3,"A:CADASTRO"
160 FIELD #3,30 AS N$
170 REM     LOOPING DE DIGITACAO E GRAVACAO
180 PRINT CHR$(12)
190 PRINT FNT$(7,20);"CHAVE : ..."
200 PRINT FNT$(9,20);"NOME  : ";STRING$(30,".
    ")
210 PRINT FNT$(7,28);
220 INPUT C
230 IF C = 0    THEN GOTO 360
240 REM         TESTA VALIDADE DA CHAVE
250 GET #3,C
260 IF N$ = "" THEN GOTO 300
270 PRINT FNT$(23,1);"JA EXISTE REGISTRO COM
    ESTA CHAVE ";
280 LINE INPUT E$
290 GOTO 180
300 REM       DIGITACAO DO NOME
310 PRINT FNT$(9,28);
320 LINE INPUT D$
330 LSET N$=D$
340 PUT #3,C
350 GOTO 180
360 REM       ENCERRAMENTO
370 PRINT CHR$(12)
380 CLOSE #3
390 END
```

José Luiz do Nascimento Silva é formado em Tecnólogo em Processamento de Dados pela PUC-RJ e trabalha como Analista de Sistemas na NABLA — Engenharia e Processamento de Dados Ltda.

Livros

"The Giant Book of Computer Software", TAB Books Inc, Editors of 73 Magazine.

"Agora que você possui um microcomputador — ou mesmo uma calculadora programável — o que fazer com ele?"

Este livro começa com esta pergunta e se propõe a respondê-la ao longo de seu texto (530 páginas) utilizando vários programas, todos eles fartamente comentados, seja através do texto ou dentro do próprio programa.

No primeiro capítulo, algumas palavras sobre linguagens: Assembler, COBOL, FORTRAN, BASIC; esta última mais detalhada. A partir do segundo capítulo, "A Suave Arte de Programação", procura-se ensinar programação através do desenvolvimento de programas de real utilidade, e não apenas didáticos. Dentre eles destaca-se um que cria um cadastro de sócios de um clube, onde a potencialidade de utilização de arquivos em sistemas com fita cassete é desenvolvida ao máximo.

Apesar do título, o livro não contém apenas software: o hard também está presente, e o último capítulo denomina-se "Combinando Hardware e Software".

Como este livro é o resultado de uma coletânea de programas desenvolvidos para as mais diversas máquinas estrangeiras, teremos que modificá-los um pouco para adaptá-los a nossos equipamentos. A elaboração destas modificações, entretanto, fica bastante facilitada pelos comentários e explicações que acompanham cada programa.

De utilidade para todos os possuidores de micros, esta coletânea é especialmente útil para aqueles que têm como hobby, além do micro, a eletrônica ou o radioamadorismo (ou ambos).

A seguir uma relação de alguns programas encontrados no livro: dissipação térmica em circuitos integrados, filtros ativos e passivos, atenuadores "T" e "Pi", cálculo de SWR, fontes de alimentação, circuitos PLL e temporizadores, uma série específica para radioamadores, programas para RTTY, programas para SSTV (Slow Scan TV), e muitos outros.

FORTRAN

Poucas instruções tornam o FORTRAN acessível para leigos na solução de problemas científicos.

Fórmulas matemáticas em programas

Orson Voerckel Galvão

Poderíamos definir FORTRAN como uma linguagem elaborada especificamente para expressar fórmulas matemáticas em programas de computadores. A expressão FORTRAN significa **FORmula TRANslator,** o que expressa bem a finalidade da linguagem.

O FORTRAN é, originalmente, um compilador desenvolvido pela IBM para a solução de problemas científicos. Por ser uma linguagem para uso por pessoas não especializadas em computação (matemáticos, físicos, estatísticos etc) ela pretende ser o mais simples possível, além de fornecer uma série de facilidades implementadas sob a forma de funções pré-declaradas. Porém, pela própria natureza da área de aplicações a que se destina, ele tem a desvantagem de não permitir grande flexibilidade no que diz respeito à entrada e saída de dados.

Outra característica interessante de FORTRAN é o fato de que a letra inicial do nome atribuído a uma variável é que vai definir o tipo de dado que esta variável vai conter. Assim, todos os nomes de variáveis que começarem com as letras **I, J, K, L, M** e **N** indicarão que o dado que ela irá conter será considerado como um número inteiro. Se for utilizada qualquer outra letra no início do nome da variável, ela será considerada uma variável do tipo real.

No entanto, existe uma instrução que permite ao usuário definir a sua própria convenção. Trata-se da instrução **IMPLICIT,** que com as definições **INTEGER, REAL, LOGICAL** e **COMPLEX,** vai fazer com que se estipulem as letras que, usadas no início do nome das variáveis, vão definir o seu tipo. Caso o usuário queira, é possível estipular ainda através destas definições todos os nomes de variáveis que deverão ser encarados de acordo com cada um destes tipos

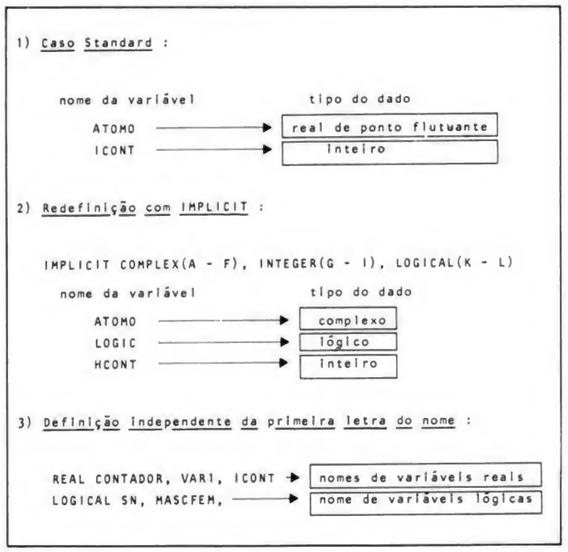


Figura 1

(inteira, real, lógica ou complexa), independentemente da primeira letra utilizada no nome das mesmas. Recapitulando através de exemplos, podemos ver os três casos na figura 1.

Devido ao grande uso de matrizes na área de ciências, o FORTRAN permite também a declaração das mesmas nos seus programas. As matrizes são definidas através da instrução **DIMENSION** e as variáveis assim definidas são chamadas de ***variáveis subscritas.*** O FORTRAN permite quantos subscritos se desejar e, em alguns casos, um dos subscritos pode ser outra variável subscrita.

COMANDOS E OPERADORES

Como é uma linguagem voltada para cálculos, o FORTRAN dispõe de poucas instruções. A maioria das linhas de programa FORTRAN vai constar de expressões aritméticas, que se utilizam de operadores +, −, *, /, **, funções e operadores lógicos.

Entre os poucos comandos mais utilizados no FORTRAN, vamos encontrar os usualmente conhecidos como comandos de controle. São eles os comandos de desvio incondicional e condicional, ambos operadores através de instruções **GOTO** e **IF,** respectivamente, e o comando **DO**, que permite o controle de *loops.*

O comando **GOTO** pode ser implementado de duas formas. A primeira das duas, a forma simples, fará com que seja realizado um desvio da sequência normal do processamento para uma determinada linha do programa. Veja um exemplo de sua utilização:

10 GOTO 201

A segunda forma de utilização do **GOTO** é chamada de ***desvio indexado.*** Neste caso, o desvio será feito para uma dentre diversas linhas especificadas, dependendo do conteúdo de uma variável inteira definida na instrução. Um exemplo da sua utilização:

100 GOTO (250, 300, 350, 999), KONT

Neste exemplo, o desvio será feito para uma das linhas especificadas entre os parênteses, dependendo do conteúdo da variável **KONT.** Se esta contiver 1, o desvio será feito para a linha 250; se contiver 2, para a linha 300; e assim por diante.

O comando **IF** foi implementado no FORTRAN de uma forma um pouco fora do usual. De uma maneira geral ele tem a forma **IF (EXP) n1, n2, n3,** onde **EXP** é uma variável ou expressão cujo conteúdo ou produto final será avaliado. A forma de avaliação se baseia no critério de tal valor ser menor, igual ou maior do que zero. Em cada um dos casos, ocorrerá um desvio para a linha especificada respectivamente por **n1, n2** e **n3.** Veja um exemplo:

100 IF (K * 3+7)91, 30, 500

Se o resultado da expressão entre parênteses for menor que zero, o desvio será feito para a linha 91; se igual a zero, para a linha 30 e, se maior que zero, para a linha 500.

Além desta forma, o **IF** poderá ser utilizado para decisões lógicas. Sob esta forma, é feita uma comparação entre dois elementos no interior dos parênteses. Se a comparação resulta em verdadeira, é executada a instrução que se segue à expressão entre parênteses (na mesma linha). Caso contrário, é encerrada a exe-

cução do **IF** e iniciada a execução da instrução encontrada na próxima linha. Veja um exemplo:

100 IF (A.LE.0) Ax = A+1

Os operadores permitidos são:

.LE. (menor ou igual)
.EQ. (igual)
.GE. (maior ou igual)
.LT. (menor que)
.GT. (maior que)
.NOT. (**NOT** lógico)
.AND. (**AND** lógico)
.OR. (**OR** lógico)

Estes operadores podem ser reunidos das mais variadas formas em uma só expressão.

O comando **DO** também vai ter uma forma de apresentação um tanto o quanto original em relação a instruções semelhantes encontradas em outras linguagens. A sua forma geral é a seguinte:

DO f i = v1, v2, v3

Pode-se ler esta instrução da seguinte maneira: *Execute a partir da próxima instrução até que* **i** *seja maior que* **v2,** *sendo que* **i** *tem um valor inicial de* **v1** *e após ser executada a instrução da linha* **f** *o valor de* **i** *deve ser incrementado com o valor encontrado em* **v3**. *Após o incremento, o processamento volta à linha seguinte à linha do* **DO.**

O fluxograma é o representado na figura 2. De uma forma geral, estas são as principais características do FORTRAN. Naturalmente existem uma série de outras instruções, tais como **CONTINUE, STOP, PAUSE, INPUT, FORMAT, READ, DATA, WRITE, DEFINE** etc. Quanto às funções, existem as mais variadas, com finalidades tais como o cálculo trigonométrico, estatística, aritmética, matrizes e por aí a fora.

Com relação ao método de programação com FORTRAN, permite-se usar extensamente sub-rotinas e subprogramas, o que lhe propicia uma certa modularidade.

Finalizando, o FORTRAN é uma linguagem que, apesar de já ser bastante antiga (em relação a algumas que são objeto de matérias neste número) parece que não entrará em desuso tão cedo. Sua simplicidade, versatilidade e variedade de funções permitem ao leigo em computação um rápido aprendizado e relativa independência quanto ao equipamento utilizado.

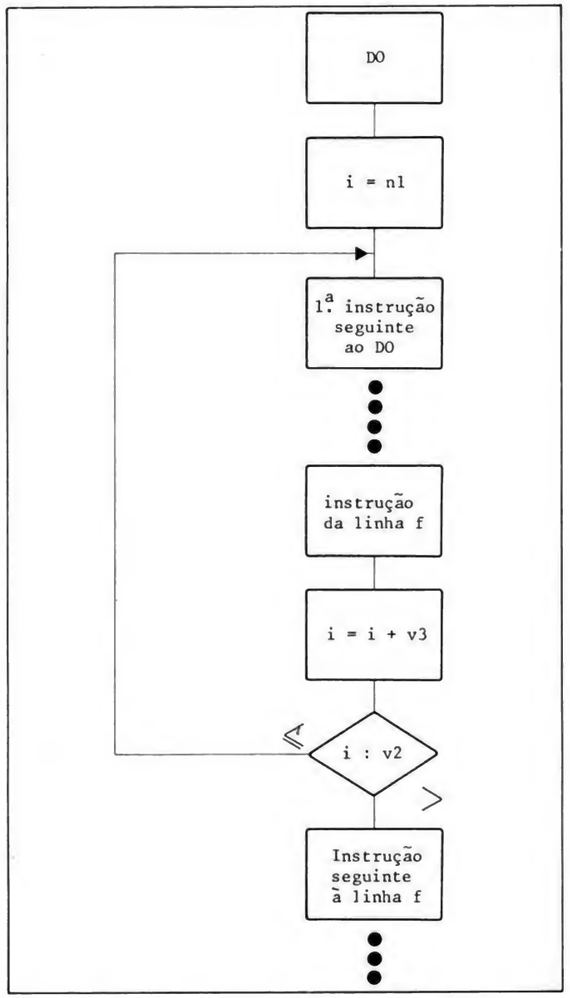


Figura 2

Orson Voerckel Galvão é Analista de Sistemas da Petrobrás Distribuidora S/A, no Rio de Janeiro, e Assessor Técnico de MICRO SISTEMAS. Orson foi o autor do Curso de BASIC publicado nos números de 2 a 9 de MICRO SISTEMAS.

MUMPS

Criada inicialmente para minis, esta linguagem vem tendo larga utilização em microcomputadores.

Trabalhando complexos de dados

Ivan Costa

MUMPS é uma linguagem que vem conquistando significativo espaço no mercado nacional de Informática e, devido às suas características, que o tornam uma alternativa extremamente atraente para os microcomputadores, pode-se prever um grande futuro para ele. Já existem empresas no país que desenvolvem software básico para MUMPS e já encontram-se disponíveis implementações deste sistema em alguns minis nacionais (por exemplo, Cobra 530 e Labo 8038) e mais recentemente em micros, como Cobra 305 e POLY 101.

MUMPS significa "Massachussets General Hospital Utility Multiprogramming System". É um sistema de programação de aplicações, caracterizado pela integração inteligente dos seguintes elementos:

- uma linguagem de alto nível com potentes recursos para manejar strings e particularmente bem dotada para aplicações que requeiram interatividade total ou parcial;
- um sistema operacional de tempo compartilhado adaptável a diferentes ambientes de hardware;
- uma base de dados dinâmica e partilhável que utiliza estruturas hierárquicas e flexíveis.

HISTÓRICO

O desenvolvimento do MUMPS teve início em 1966, no Laboratório de Ciência de Computação do Massachussets General Hospital, em Boston, EUA, em resposta às necessidades computacionais decorrentes das rotinas e das técnicas de administração hospitalar, bem como aquelas associadas às experimentações e técnicas médicas. Deste modo, os critérios que nortearam tal projeto incluíram a flexibilidade na interface com o ambiente, capacidade de manipulação de textos, organização dos dados numa estrutura de acesso rápido e simples em ambiente multiusuário e partilhável e, finalmente, a existência de uma linguagem de alto nível, voltada principalmente para o desenvolvimento de sistemas de aplicações.

O resultado final integrou, num só sistema, um programa de controle de tempo partilhado, uma linguagem de alto nível e de fácil manuseio e uma base hierárquica de dados cuja criação e acesso são transparentes ao usuário. Esta estrutura de dados, por sua utilização transparente e fácil transposição para a realidade lógica, tornou-a extremamente adequada para o manuseio de complexos de dados dentro do ambiente para o qual o sistema foi projetado.

Em 1972, o National Center for Health Services Research e o National Bureau of Standards determinaram a criação do MUMPS Development Committee para estudar a viabilidade de se estabelecer um MUMPS padrão que fosse utilizado tanto por usuários como por fabricantes. Após alguns anos de esforço, um padrão foi adotado e suas especificações foram submetidas para aprovação ao American National Standard Institute, finalmente concedida em 1977.

A LINGUAGEM MUMPS

A linguagem MUMPS foi projetada para ser uma linguagem de fácil aprendizado, com métodos simples de criação, modificação e depuração de programas. A primeira característica a ser considerada em sua análise é o fato dela ser interpretada, diferente da maioria das linguagens convencionais que são compiladas. Em uma linguagem interpretada, os programas são executados diretamente do código-fonte, não sendo necessário compilar ou traduzir o programa para outra forma antes de executá-lo. Este fator pode representar significativa economia de tempo para o programador. Além disso, trabalhar somente na forma simbólica facilita o desenvolvimento e a manutenção de programas. Na fase de depuração, erros podem ser analisados e rapidamente corrigidos em pequenos segmentos de programa, em processo interativo com um terminal.

Outra característica importante do MUMPS, que a diferencia das demais linguagens, é o fato de possuir, embutida, uma estrutura de dados hierárquica, permitindo acessos simultâneos, com segurança, a uma base de dados compartilhada. Este recurso a torna similar a alguns sistemas de gerenciamento de base de dados. Em adição, na maioria das vezes, a linguagem MUMPS é usada sob seu próprio sistema operacional, aumentando assim sua eficiência de execução.

Um programa MUMPS é extremamente simples de codificar e dispensa por completo o uso de formulários especiais. Sua estrutura é baseada em linhas, que podem ser identificadas por um rótulo, e onde estão representados comandos, funções, variáveis, operadores e expressões de maneira compacta e integrada.

Como pontos característicos da linguagem, podemos ressaltar:

- conjuntos de comandos envolvendo operações de atribuição, desvio, iterações etc. Todos eles podem ser abreviados usando-se somente a primeira letra (economia de tempo e espaço) e na sua maioria podem ser adjetivados com pós-condicionamento e temporização (veja no quadro, os comandos definidos no MUMPS padrão);
- vários comandos podem aparecer na mesma linha. O MUMPS exige apenas um separador sintático (o caráter branco) entre comandos e seus argumentos e entre comandos e outros comandos. Um comando condicional valorizado como falso, entretanto, fará com que o resto dos comandos naquela linha sejam ignorados;
- uma linha pode ser direta, caso em que é imediatamente executada após a sua entrada em terminal, ou indireta, sendo armazenada para ser executada como parte de um programa;
- em extensão aos operadores aritméticos e lógicos convencionais, o MUMPS possui uma série de operadores e funções para tratamento de cadeias de caracteres ou informações sob a forma de texto;
- um programa pode realizar diversas operações entre cadeias de caracteres, tais como comparações, concatenações, extrações de subcadeias, pesquisa de ocorrência, obtenção de tamanho etc. Estes recursos facilitam operações de crítica, tal como mostra a figura 1, que apresenta um trecho de um programa MUMPS para leitura e crítica de um número de seis caracteres digitados no terminal. O valor digitado é armazenado na variável **M** e uma crítica é realizada para verificar se **M** está na forma correta, ou seja, três dígitos seguidos de um hífen e mais dois dígitos. Valores incorretos causam a exibição de uma mensagem de erro e a requisição de um novo valor. O comando **ZPRINT** lista o conteúdo da partição e o comando **DO** causa a execução que está ilustrada (as entradas do usuário estão sublinhadas para distingui-las das mensagens do sistema);
- os usuários MUMPS operam dentro de partições residentes em memória, utilizando variáveis locais (exclusivas do usuário que as criou), variáveis globais (armazenadas em disco e compartilhadas por todos os usuários) e variáveis especiais (consultadas pelo usuário e mantidas pelo sistema).

Os recentes avanços na tecnologia de microprocessadores, com a consequente redução dos seus preços, viabilizou seu uso em diversas aplicações, tais como: sistemas domésticos, automação de pequenos »

Descrição dos comandos MUMPS

BREAK – fornece pontos de acesso, internos ao MUMPS, para fins de depuração.
CLOSE – devolve uma unidade de E/S ao sistema, após sua utilização por um usuário.
DO – chamadas de rotinas (internas ou externas).
ELSE – execução condicional (executa o restante da linha se o valor da variável especial **$T** é falso).
FOR – iteração ou execução repetida de um segmento específico de comandos.
GOTO – desvio incondicional para determinada linha de uma rotina (interna ou externa).
HALT – término de execução da tarefa atual na partição em questão.
HANG – interrupção do processamento por determinado tempo especificado no comando.
IF – execução condicional (executa o restante da linha se o valor da variável especial **$T** é verdadeiro).
KILL – elimina variáveis (locais ou globais).
LOCK – aloca uma global a um determinado usuário.
OPEN – aloca uma unidade de E/S a um determinado usuário.
QUIT – término de execução de um comando **FOR, DO** ou **XECUTE.**
READ – comando para entrada de dados de um dispositivo de E/S.
SET – atribuição de valores a variáveis (locais ou globais).
USE – designa uma unidade de E/S como dispositivo corrente.
VIEW – fornece pontos de acesso, internos ao MUMPS, para exame de informações do sistema.
WRITE – comando de saída de dados para um dispositivo de E/S.
XECUTE – permite interpretar seus argumentos como uma linha de código MUMPS naquele ponto do programa.

```
ZPRINT

10 READ !,"MATRICULA: ",M
   IF M'?3N1"-"2N W "  MATRICULA INVALIDA" GOTO 10

DO 10

MATRICULA: 12-56  MATRICULA INVALIDA
MATRICULA: 123-2  MATRICULA INVALIDA
MATRICULA: 1234-2  MATRICULA INVALIDA
MATRICULA: 123-22
```

Figura 1 — Trecho de um programa MUMPS para leitura e crítica de um número de seis caracteres.

escritórios, dedicação exclusiva em centros de pesquisas etc. Todas essas aplicações têm exigido software específico para o ambiente de micros.

Projetado inicialmente para minicomputadores, o MUMPS revelou-se adequado para uso em micros, sem alterações significativas de seu desempenho relativo. Os recursos disponíveis no MUMPS para tratamento de strings e gerenciamento de arquivos, o tornam ideal para algumas aplicações típicas de micros, tais como: processamento de textos, sistemas de ensino e pequenas aplicações pessoais e comerciais. Além disso, o MUMPS possui características não facilmente encontradas em outras linguagens voltadas para micros, incluindo capacidade para comunicação de dados e possibilidade de operação em ambiente multiusuário.

Uma outra vantagem do MUMPS é o fato da linguagem ser padronizada, o que a torna uma poderosa ferramenta de ligação entre micros e minis, ou sistemas de grande porte, em sistemas hierárquicos (distribuídos ou não). Esta característica é muito importante, pois se por um lado, em termos de sistema operacional em micros, existe uma tendência para a padronização (CP/M, por exemplo), em relação às linguagens não se conseguiu até hoje uma padronização aceitável. Para constatar isto, basta observar, por exemplo, a grande diversidade de formas de BASIC existentes no mercado.

Quando consideramos a compatibilidade de software entre micros, minis e sistemas de grande porte, a situação tende a agravar-se. Compiladores de sistemas maiores são implementados usualmente em micros, com sérias limitações, tornando-se apenas "subsets" do produto original. Em contrapartida, os recursos de algumas das melhores linguagens para micros (Pascal, por exemplo) não se encontram disponíveis em sistemas maiores.

Desenvolvidas no país, existem versões MUMPS disponíveis para o Z80 e Intel 8080/8085. Além disso, uma quantidade razoável de software de aplicação pode ser encontrada em casas especializadas.

Podemos citar como principais causas para a lenta expansão do MUMPS a pouca divulgação da linguagem, receio de mudar de uma linguagem tradicional para outra, de nova filosofia, falta de compatibilidade do MUMPS com as demais linguagens e dificuldade de conversão de programas já existentes em outras linguagens para o MUMPS.

Naturalmente, como toda linguagem (e sistema operacional), o MUMPS possui suas limitações e em determinadas aplicações (em sistemas puramente "Batch", por exemplo) ele não será uma boa escolha. Mas se considerarmos todo o potencial em termos de recursos técnicos, facilidade de aprendizado, simplicidade de operação, portabilidade (características importantes, principalmente para o usuário de micros que não dispõe de equipes caras para resolver seus problemas), e somarmos a tudo isto uma ótima relação custo/desempenho, poderemos estar encontrando uma solução simples e eficaz para nossos problemas: a solução MUMPS.

Ivan Costa é Engenheiro de Sistemas e Computação desde 1980, formado pela UERJ. Atualmente é professor de Técnicas de Programação nas Faculdades Integradas Estácio de Sá e técnico da diretoria de PD do IPLAN RIO, responsável pela manutenção do sistema operacional MUMPS.

```
MUMPS                              BASIC

10 READ "N=",N                     0010 INPUT "N=",N
   SET C=1,S=0                     0020 LET   C=1
30 READ "V=",V                     0030 INPUT "V=",V
   SET S=S+V                       0040 LET   S=S+V
   SET C=C+1                       0050 LET   C=C+1
   IF C'>N GOTO 30                 0060 IF    C<=N THEN GOTO 30
   WRITE "S=",S                    0070 PRINT "S=";S
   QUIT                            0090 END

   OU

10 R "N=",N S C=1,S=0
30 R "V=",V S S=S+V,C=C+1 G:C'>N 30 W "S=",S Q
```

Figura 2 – Comparação de programas MUMPS e BASIC. O MUMPS está codificado de duas formas diferentes para evidenciar o quanto ele pode ser compacto.

LOGO

Mais do que uma linguagem, Logo é um elo entre a matemática, o computador e o pensamento lógico.

A matemática da tartaruga

Dulce Madalena Autran von Pfuhl

A crescente difusão dos computadores pessoais — cada vez mais acessíveis — coloca uma questão muito séria, apesar de óbvia: o que fazer com eles? Os fabricantes procuram inventar recheios atraentes para suas máquinas: linguagens científicas e comerciais, editores de texto, joguinhos e... o que mais? Todo esforço é feito para ampliar as aplicações e atingir um número maior de usuários.

Várias pessoas que desde o curso primário foram rotuladas como "competentes em matemática" têm interesse natural por computadores, e procuram saber como usá-los. Mas, e os numerosos "inimigos da matemática", os que desde as primeiras frustrantes experiências com a matemática escolar se declararam "incompetentes" e agiram de acordo? Esses têm medo dos computadores, ou simplesmente não acreditam que eles possam proporcionar coisas interessantes para fazer. Afinal, não se pode lidar com um computador se não se sabe e gosta de matemática, certo?

Errado. No Laboratório Logo os computadores são usados por crianças, adolescentes e adultos de todos os níveis de interesse matemático, com enorme sucesso. Qualquer estudante é capaz de, em pouco tempo, fazer seus próprios programas, originais e sofisticados, tirando dessa atividade uma grande satisfação pessoal.

Neste artigo vamos tratar de Logo — que é o nome de uma filosofia de educação e também de uma família de linguagens de programação em constante desenvolvimento que tornam possível a realização dessa filosofia.

Logo usa o computador como ferramenta para ajudar a aprender, a jogar, a brincar, a experimentar. Ao contrário das atividades de instrução programada, onde o estudante é submetido a um programa que o ajuda a estudar, avaliando suas respostas e detectando os pontos fracos do aprendizado, as atividades no Laboratório Logo colocam os recursos do computador sob o comando do estudante, que assume o controle do equipamento, cria seus próprios objetivos e os realiza. Em Logo, a pessoa programa o computador, ao invés de ser programada por ele.

Logo é uma tentativa de desenvolver um novo método, não de ensinar matemática, e sim de ensinar a pensar — e para isso a matemática pode ser um bom meio. As crianças não vêm aprender Logo com as cabeças vazias; apesar de terem vivido pouco tempo, elas já pensaram, já resolveram inúmeros problemas, já fizeram experiências com suas idéias. Essa é a parte vital de cada pessoa: a parte que imagina sobre o mundo e reage ao mundo ativa e individualmente, e a parte que se desenvolve no ambiente Logo de aprendizado. Aparece na criança um início de um auto-conceito de pensador e criador — o que resulta numa visão positiva do mundo.

APRENDENDO POR ANALOGIA

Algumas idéias ocorrem de maneira tão forte e significativa que nos acompanham pelo resto da vida e se aplicam a diferentes contextos. Nas ligações entre matemática, computadores e a vida cotidiana, via Logo, várias dessas idéias são enfatizadas. Como exemplo, vamos usar a metáfora do antropomorfismo (crença ou doutrina que atribui a Deus ou a deuses formas ou atributos humanos) para alguns conceitos de matemática e computação, como são apresentados às crianças no Laboratório Logo.

Variáveis podem ser ilustradas como homenzinhos, cada um possuindo um nome e uma coisa (que em geral é um número ou uma palavra). Muitas idéias computacionais podem ser colocadas no "modelo dos homenzinhos" de como funciona um computador: a unidade aritmética e lógica é um grupo de matemáticos, a memória é uma biblioteca, administrada pela bibliotecária, onde são guardados programas e dados. Mensagens são passadas através de fios de telefone entre as várias unidades. A execução de um procedimento é conduzida por um líder que, no caso de haver um subprocedimento, passa o comando ao líder correspondente, recebendo-o de volta quando o subprocedimento termina. Essa analogia leva à análise e ao entendimento do processo de funcionamento do computador, sem necessidade de conhecimentos de hardware.

Uma atividade interessante é simular o computador, sugerindo que cada criança faça um dos papéis, executando procedimentos. Pode-se também brincar de tartaruga. Uma das crianças é vendada e não sabe que desenho vai fazer, mas só pode andar em linha reta e girar sem sair do lugar. Os outros devem dar os comandos para que ela crie um desenho no chão. Esse jogo facilita a compreensão dos números e as crianças sentem a necessidade de um sistema de descrição preciso.

Quando a criança começa a observar as relações entre os tipos de coisas que ela aprende brincando e os tipos de idéias que aparecem nas experiências com computadores, ela começa a entender a universalidade das idéias da matemática e da resolução de problemas.

PRINCIPAIS RECURSOS

Aqui descreveremos, resumidamente, a versão do MIT LOGO implementada no Apple II (para uma descrição completa, consulte o manual de linguagens escrito por Harold Abelson). Essa versão requer um floppy disk drive e 48 Kb de memória, com uma extensão de 16 Kb, como o Apple Language Card. Além do disquete Logo, o usuário necessita apenas de disquetes para guardar os programas que desejar no sistema de arquivo.

Apesar de ser muito fácil de aprender, Logo é também um sistema de grande poder e abrangência de aplicações. Um programa Logo é constituído de comandos agrupados na forma de procedimentos. Cada comando pode ser, ou um comando primitivo do sistema, ou qualquer outro procedimento definido pelo usuário, de modo que procedimentos são formados de procedimentos etc., até um nível arbitrário de complexidade. Os procedimentos podem se comunicar através de entradas e saídas (inputs e outputs).

Um comando ou procedimento é executado simplesmente teclando-o no console, pois o editor permite definir, executar e modificar procedimentos sem preocupação com compiladores, carregadores, monitores etc.

Logo trabalha com números, cadeias de caracteres e listas. Lista é uma sequência de itens, onde cada item pode ser número, cadeia de caracteres ou lista. As listas, portanto, podem ser formadas de listas, de listas de listas etc., até um nível arbitrário de complexidade.

Estas características tornam Logo muito flexível e conveniente para todos os graus de sofisticação das aplicações.

A tartaruga é um pequeno triângulo que aparece no centro do vídeo quando se tecla o comando **DRAW** (desenho). Ela pode se movimentar pela tela por meio de comandos, deixando um traço por onde passa. O comando **FORWARD** (para a frente) faz com que ela se mova na direção que está apontando, e **BACK** (para trás) na direção oposta. Estes comandos devem ser seguidos de um número que diz quantas unidades ela deve caminhar. Os comandos **RIGHT** (direita) e **LEFT** (esquerda) fazem com que ela gire sem sair do mesmo ponto, e devem ser seguidos do número de graus da rotação desejada. Os números que seguem os comandos chamam-se inputs (entradas).

Para mover a tartaruga sem desenhar, dá-se o comando **PENUP** (levantar a caneta), o qual não requer inputs. Os subsequentes **FORWARD** e **BACK** vão fazer a tartaruga se mover sem deixar traço. Para voltar a desenhar, usa-se **PENDOWN** (abaixar a caneta). O comando **HIDETURTLE** (esconder a tartaruga) faz com que o triângulo indicador da posição da tartaruga desapareça, embora ela continue no mesmo lugar e possa obedecer a todos os comandos. **SHOWTURTLE** (aparecer a tartaruga) faz o triângulo reaparecer. O

As experiências no MIT

O criador da filosofia Logo, prof. Seymour Papert, do MIT (Massachusetts Institute of Technology), compara o aprendizado de linguagens de programação ao ensino de línguas estrangeiras: nas escolas, o ensino de francês é penoso e improdutivo, enquanto aprender francês na França é fácil e rápido. Por que não considerar o computador como um ser que fala matemática, e construir um computador com o qual as pessoas queiram e gostem de se comunicar, mesmo as "ruins em matemática"?

O grupo de pesquisas do Laboratório Logo do MIT construiu um ambiente de aprendizado com a presença de computadores que controlam vários tipos de terminais. Há um robô — chamado tartaruga — que passeia sobre um papel no chão e tem uma caneta que deixa um traço por onde passa. É possível fazer os mais variados desenhos desse modo: estacionar a tartaruga entre dois blocos, fazê-la percorrer um labirinto, ou muitos outros jogos.

A tartaruga só se move através de comandos dados pelo computador. Há um vídeo colorido onde uma outra versão da tartaruga, um pequeno triângulo luminoso passeia pela tela, deixando um rastro, como o robô no chão. Há uma outra caixa de música na qual se pode programar o que se quer tocar, há uma caixa de vozes onde se constroem palavras e frases sintéticas. E assim existe motivação para que qualquer pessoa deseje fazer funcionar tantas atrações.

Além dos aparelhos, o ambiente Logo se compõe também de jogos interessantes, atividades físicas, como andar de pernas de pau, fazer malabarismos (manter três ou mais objetos no ar, em movimento, jogando e aparando), andar de monociclo (bicicleta de uma roda só) etc.

A sala de aula é compartilhada por todos, livremente, novatos e entendidos; as paredes são forradas de sugestões dadas por todos os participantes, todos trocam idéias, ensinam-se uns aos outros, emprestam-se programas; grupos se reúnem para programar ou discutir em conjunto, fazer comentários, elaborar e executar projetos.

Finalmente, para cada grupo de duas ou três crianças que frequentam o Laboratório Logo há um instrutor responsável. Ele dá um mínimo de informações básicas para o uso dos terminais e está à disposição para fornecer mais instruções de acordo com o progresso e a solicitação de cada aluno ou do grupo.

comando **DRAW** apaga os traços na tela e recoloca a tartaruga no centro, apontando para cima.

Há abreviações para todos os comandos: **FD** para **FORWARD, BK** para **BACK, RT** para **RIGHT, LT** para **LEFT, PU** para **PENUP, PD** para **PENDOWN, HD** para **HIDETURTLE** e **ST** para **SHOWTURTLE.**

O EXEMPLO DO QUADRADO

Um dos desenhos mais populares para os iniciantes é o quadrado:

```
FORWARD  100
RIGHT  90
FORWARD  100
RIGHT  90
FORWARD 100
RIGHT  90
FORWARD  100
```

Pode-se editar (corrigir) linhas Logo inserindo ou apagando caracteres na posição em que está o cursor (um sinal que pisca constantemente). Alguns comandos podem movimentar o cursor por todo o texto.

Com o comando **REPEAT** podemos repetir um certo grupo de comandos quantas vezes desejarmos através de dois inputs: um número e uma lista. Portanto, o quadrado ficaria assim:

REPEAT 4 FORWARD 100 RIGHT 90.

Se o usuário tem como monitor uma TV a cores, pode usar os comandos **PENCOLOR** (cor da caneta) que muda a cor dos traços, e o **BACKGROUND** (fundo) que muda a cor da tela. Há sete opções: preto (0), branco (1), verde (2), violeta (3), laranja (4) e azul (5). **PENCOLOR 6** inverte a cor dos pontos pelos quais a tartaruga passa.

Os comandos **FORWARD, DRAW** etc. podem ser vistos como "palavras que Logo já sabe", ou primitivas. Em Logo é fácil incorporar novas palavras ao vocabulário do sistema — em procedimentos definidos pelo usuário podem ser usados como se fossem primitivas. Palavras novas são definidas em termos de palavras já conhecidas. Se o programador desejar que a sequência de comandos que desenha aquele mesmo quadrado seja incorporada ao vocabulário, basta escolher um nome para ela e definir:

```
TO QUADRADO
REPEAT  4  [FORWARD 100  RIGHT 90]
END
```

A linha título tem a palavra **TO** (para) e o nome escolhido. Após teclar o **END** (fim), o procedimento está definido, e Logo reconhecerá a palavra **QUADRADO.** Ela agora pode ser executada ou fazer parte de outros procedimentos. Exemplo:

```
TO JANELA
QUADRADO
RIGHT  90
QUADRADO
END
```

É bom lembrar que procedimentos podem ser editados por meio do comando **EDIT** tendo como input o nome do procedimento.

Assim como os comandos **FORWARD 100** e **FORWARD 50** desenham coisas diferentes, podemos definir procedimentos que aceitam inputs. Voltando ao exemplo do quadrado, poderíamos definir:

```
TO QUADRADO  :LADO
REPEAT  4  [FORWARD  :LADO  RIGHT 90]
END
```

e desenhar quadrados de tamanhos diferentes: **QUADRADO 100, QUADRADO 50** etc. A cada execução, o input que segue **QUADRADO** é passado como valor de **LADO.**

Os procedimentos podem ter mais de um input:

```
TO RETANGULO  :BASE  :ALTURA
FORWARD  :ALTURA
RIGHT  90
FORWARD  :BASE
RIGHT  90
FORWARD  :ALTURA
RIGHT  90
FORWARD  :BASE
END
```

Podemos ainda incluir num procedimento uma chamada a si mesmo, ou seja, um dos comandos do procedimento é seu próprio nome. Vejamos:

```
TO QUADRADO :LADO
FORWARD :LADO
RIGHT 90
QUADRADO :LADO
END
```

A execução deste procedimento faz a tartaruga desenhar o quadrado e continuar caminhando sobre o mesmo quadrado indefinidamente, pois cada execução de **QUADRADO** inclui: andar para frente, girar de 90 graus e executar **QUADRADO.** Para desenhar um quadrado, esse processo não é útil, mas o procedimento **POLIGONO** desenha um polígono com um número qualquer de lados: »

```
TO POLIGONO :LADO :ANGULO
FORWARD :LADO
RIGHT :ANGULO
POLIGONO :LADO :ANGULO
END
```

Podemos modificar um pouco o procedimento e ter:

```
TO POLISPI :LADO :ANGULO
FORWARD :LADO
RIGHT :ANGULO
POLISPI (:LADO + 5) :ANGULO
END
```

que produz uma espiral poligonal. Dependendo dos inputs, os procedimentos **POLIGONO** e **POLISP** podem produzir desenhos variadíssimos e inesperados.

Usar um procedimento como parte da definição do próprio procedimento chama-se **recursão.** O comando **PRINT** (imprima), que não faz parte dos comandos geométricos, imprime na tela o valor de seu input. Assim, **PRINT 5** imprime **5** e **PRINT: A** imprime o valor de A. Consideremos o procedimento recursivo.

```
TO CONT :NUMERO
PRINT :NUMERO
CONT :NUMERO - 1
END
```

Se usarmos o comando **CONT 5,** Logo vai imprimir **5 4 3 2 1 0 -1 -2** ... até interrompermos a execução. O procedimento que imprime até 1 e pára é:

```
TO CONT :NUMERO
IF :NUMERO= 0 STOP
PRINT :NUMERO
CONT :NUMERO - 1
END
```

O comando **IF** (se) é usado para fazer testes, que dão como resultado **TRUE** (verdadeiro) ou **FALSE** (falso). Se o resultado for **TRUE** (no caso, se o valor de **NÚMERO** é zero), o comando que está na mesma linha é executado — no caso, **STOP** (pare). Se o resultado é **FALSE** a execução do procedimento prossegue no início da próxima linha.

O comando **STOP** pára o procedimento ao qual pertence. Se este procedimento foi chamado por um outro, este outro continua a ser executado. Por exemplo:

```
TO CONTAR :DADO
CONT :DADO
PRINT 1000
END
```

e então, ao executarmos **CONTAR 3,** teremos impresso **3 2 1 1000.**

Agora pense: o que fará o procedimento **SURPRESA?**

```
TO SURPRESA :NUMERO
IF :NUMERO= 0 STOP
SURPRESA :NUMERO - 1
PRINT :NUMERO
END
```

NÚMEROS E PALAVRAS

Logo possui operações de adição, subtração, multiplicação e divisão, permitindo operações com inteiros entre -2^{31} e 2^{31}, e decimais de valor absoluto entre 10^{38} e 10^{-38} aproximadamente. As operações com decimais são corretas até sete dígitos significativos e a notação exponencial é permitida. A divisão é sempre feita com decimais, mas existem as positivas **QUOTIENT** (quociente) e **REMAINDER** (resto). É possível arredondar um decimal para o inteiro mais próximo por meio de **ROUND.** Existe ainda a operação **RANDOM** (aleatório) que toma um inteiro positivo **n** como input e dá um inteiro qualquer entre **0** e **n-1.**

Usando operações aritméticas, podemos escrever procedimentos que manipulam números:

```
TO PQUAD :X
PRINT :X * :X
END
```

imprime o quadrado de seu input, enquanto

```
TO PMED :X :Y
PRINT (:X + :Y)/2
END
```

imprime a média dos dois **inputs.** Esses procedimentos não têm grande utilidade, pois não há mais nada que possamos fazer com eles. A importância dos procedimentos está em podermos usá-los como comandos em outros procedimentos. No entanto não podemos, por exemplo, combinar **PMED** e **PQUAD** para fazer o quadrado da média de dois números. É necessário que um procedimento produza um resultado acessível a outros procedimentos, e isso é obtido em Logo pelo comando **OUTPUT.** O procedimento **QUAD** usa **OUTPUT** da seguinte forma:

```
TO QUAD :X
OUTPUT :X * :X
END
```

e quando é executado, fornece uma saída para o comando que o chamou:

```
PRINT QUAD 3    imprime 9
PRINT (QUAD 3  QUAD 4)    imprime 25
```


O mesmo pode ser feito com a média:

```
TO MED :X :Y
OUTPUT (:X + :Y)/2
END
```

e agora podemos combinar os dois procedimentos:

```
PRINT QUAD (MED 5 6) imprime 30.25
PRINT MED (QUAD 5) (QUAD 6)  imprime 30.5
```

e formar novos procedimentos:

```
TO MEDIA.DOS.QUADRADOS  :X :Y
OUTPUT MED (QUAD :X) (QUAD :Y)
END
```

A saída de um procedimento, portanto, serve de entrada para outro.

Um procedimento para dar o valor absoluto de um número seria:

```
TO ABS :X
IF :X<0  OUTPUT (-:X)
OUTPUT :X
END
```

e para exponenciação, temos um procedimento recursivo:

```
TO EXP :X :N
IF :N=1  OUTPUT :X
OUTPUT :X* EXP :X (:N - 1)
END
```

, que pode ser melhorado para

```
TO EXP  :X  :N
IF :N=1 OUTPUT  :X
IF (PAR? :N) OUTPUT QUAD (EXP :X QUOTIENT :N 2)
OUTPUT :X* EXP :X (:N-1)
END
```

pois para calcular X^{100} fazemos cem multiplicações com o primeiro e apenas nove com o segundo.

Logo também trabalha com cadeias de caracteres, que são chamadas palavras. Há operações para combinar palavras em palavras mais longas ou então para separá-las em partes. Palavras podem ser usadas como entradas e saídas de procedimentos. Para indicar uma palavra em Logo, esta deve vir precedida por aspas. Assim, **PRINT "OI!** imprime **OI!; PRINT "2+3** im-

prime **2 +3;** mas **PRINT 2 +3** imprime **5.**

Vejamos as operações com palavras. **FIRST** (primeiro) tem como output o primeiro caráter do seu input; **LAST** (último) tem como output o último caráter de seu input; **BUTFIRST** (exceto o primeiro) tem como output uma palavra que contém o input todo exceto o primeiro caráter; **BUTLAST** (exceto o último) tem como output uma palavra que contém o input todo, exceto o último caráter. Exemplificando:

```
PRINT FIRST "ABCD    imprime A
PRINT BUTFIRST "ABCD    imprime BCD
PRINT LAST (BUTLAST "ABCD)    imprime C
PRINT BUTFIRST "A    imprime    (em branco)
```

No último exemplo foi impresso o que restou de **"A** quando retirado o **A**, ou seja, a palavra sem caracteres — é o que se chama "palavra vazia". Pode-se usar a palavra vazia como input: **PRINT"** imprime (em branco).

A operação **WORD** (palavra) tem duas palavras como **input** e combina as duas numa só, isto é, **PRINT WORD "GUARDA"CHUVA** imprime **GUARDACHUVA.** Tais operações com palavras também se aplicam a números:

```
PRINT FIRST 123  imprime  1
PRINT WORD 12  34  imprime  1234
PRINT (WORD 12  34)+(WORD 56  78)  imprime  6912
```

Uma das vantagens de Logo é permitir a manipulação de sequências de palavras não obrigatoriamente caráter por caráter, mas também palavra por palavra. Uma lista é uma sequência de itens entre colchetes, e a separação entre eles é feita com espaços em branco. As palavras na lista não têm aspas e os colchetes não são impressos. Como podemos ver, **PRINT [ISTO E' UMA LISTA]** imprime **ISTO E' UMA LISTA.**

As operações **FIRST, BUTFIRST, LAST** e **BUTLAST** também valem para listas, ou seja,

```
PRINT FIRST  [ISTO E' UMA LISTA]  imprime  ISTO
PRINT BUTLAST  [ADORO CACHORRO QUENTE]  imprime  ADORO CACHORRO
PRINT BUTFIRST  [LISTA]  imprime  (em branco).
```

Podemos notar que **BUTFIRST** é uma lista sem elementos, ou linha vazia, que pode ser usada em Logo como **[ ]** (colchetes) e não é igual à palavra vazia Uma lista nunca é igual a uma palavra, mesmo que a impressão seja igual. **PRINT "SIM** imprime **SIM** e **PRINT [SIM]** imprime **SIM** mas **PRINT "SIM =[SIM]** imprime **FALSE.**

A operação **SENTENCE** para listas é análoga a **WORD** para palavras. Vejamos alguns exemplos:

```
PRINT SENTENCE  [ISTO E']  [UMA LISTA]  imprime  ISTO E' UMA LISTA
PRINT SENTENCE  "ISTO  [E' UMA LISTA]  imprime  ISTO E' UMA LISTA
PRINT SENTENCE  "ISTO  "TAMBEM  imprime  ISTO TAMBEM
```

Já vimos nomes que se referem a inputs e nomes de procedimentos. Mas ainda há nomes que podemos atribuir livremente a coisas. O comando **MAKE** (faça) associa seus dois inputs da seguinte forma:

```
MAKE  "NUMERO  77
PRINT  :NUMERO  imprime  77
```

O primeiro input de **MAKE** é o nome, e o segundo é a coisa associada ao nome. No exemplo anterior estamos dando o nome **NUMERO** ao valor **77.** No comando **PRINT : NUMERO** vemos que **:** (dois pontos) fornece a coisa associada ao nome. Este sinal é uma abreviação do comando **THING.** Outros exemplos:

```
MAKE  "COR  "VERDE
PRINT  :COR  imprime  VERDE
MAKE  "FRASE.  [JA' VEM CHUVA]
PRINT :FRASE  imprime  JA' VEM CHUVA
MAKE  (WORD  "N 5)  [J  K  L]
PRINT  :N5  imprime J  K  L
```

LOGO NO BRASIL

A Universidade Estadual de Campinas conta com um pequeno grupo de pesquisa em atividades Logo. Dispomos de uma versão mais antiga da linguagem, implementada em um PDP-10 com terminal gráfico OT-40, e um dos pesquisadores do grupo, o prof. Djalma Salles, construiu um protótipo da tartaruga de solo.

Dulce Madalena Autran von Pfuhl é formada em Física pela Universidade de São Paulo e mestre em Ciências pelo Massachusetts Institute of Technology (MIT), EUA. Trabalhou no Laboratório Logo do MIT com o prof. Seymour Papert, e atualmente faz parte do corpo docente da Universidade Estadual de Campinas.

LTD

Aqui vai uma idéia sobre o funcionamento da linguagem LTD, utilizada pelos equipamentos da Cobra.

Um mecanismo de entrada e crítica de dados

Nilton do Valle Oliveira

A Linguagem de Transcrição de Dados — LTD é uma linguagem de alto nível, projetada para atender às necessidades do usuário na fase de entrada de dados, ou seja, para permitir um mecanismo automático e simultâneo de entrada e crítica de dados.

Ainda que o método mais utilizado seja a perfuração de cartões, neste os erros só podem ser detectados em fases posteriores. O compilador LTD, por sua vez, traduz as instruções fontes em segmentos de códigos identificáveis por um dos interpretadores conforme a fase que estiver sendo executada: 1) fase de crítica **(.INTER)**; 2) fase de transferência **(.REFOR).**

IDÉIA DE UM PROGRAMA LTD

Um programa escrito em LTD tem por objetivo:

- a crítica de dados referentes a um determinado formulário, ou
- a transferência de dados de um arquivo gerado por um programa de crítica para um dispositivo designado por uma unidade lógica, ou
- a criação de tabelas externas que poderão ser modificadas sem que haja a necessidade de recompilação dos programas que as utilizam.

Sendo assim, a linguagem apresenta certas instruções que só podem ser utilizadas em uma destas três fases. De um modo geral, um programa escrito em LTD apresenta o aspecto tal como mostra a figura 1. Vejamos então como pode ficar a nível de formato **(FORM).** Funções e tabelas poderão ser declaradas. Exemplos:

```
1.1  PROG   CRITO1   (FP=05);
1.2  TRANSF  CRITO2  (MG=F,
     BL=1000, REG=100, IT=88);
```

No primeiro exemplo (1.1), a palavra reservada **PROG** indica que se trata de um programa de crítica identificado por **CRIT01,** cujo formato de partida é igual ao de número **05.**

No segundo exemplo (1.2), a palavra reservada **TRANSF** indica que se trata de um programa de transferência identificado por **CRIT02,** contendo as seguintes opções de transferência:

— **MG:** indica a organização dos registros de saída (fixo, variável, indefinido/blocado, não blocado);

— **BL:** tamanho do bloco a ser transferido;
— **REG:** tamanho do registro lógico a ser transferido;
— **IT:** indica o número do formato que deve ser executado no início da transferência.

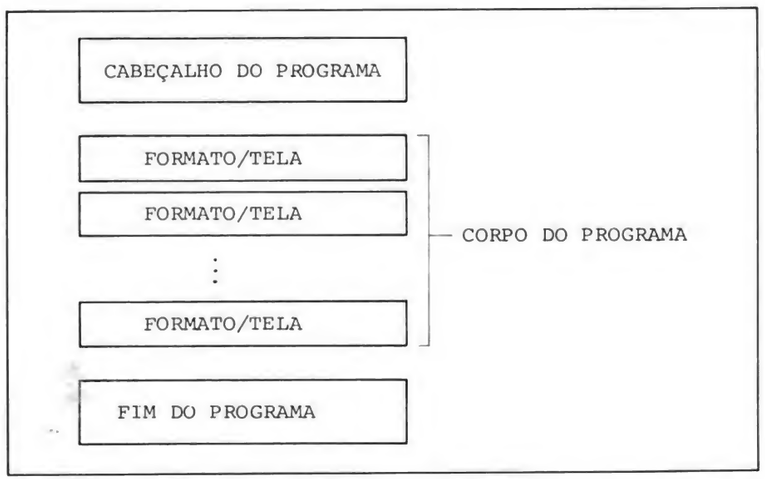


Figura 1 — Formato de um programa LTD

1. CABEÇALHO DO PROGRAMA

É a parte do programa onde é possível informar se este é de crítica **(PROG)** ou de transferência **(TRANSF).** Apresenta também opções de processamento que, no caso de um programa de crítica, operam cada uma de suas três partes.

2. FORMATO

Um formato deve ser iniciado pela palavra reservada **FORM** seguida de um número sem sinal que poderá variar de 0 a 99. Deve ser finalizado por uma das instruções:

SEG[nn]; *ou* SEG[nn];

onde **nn** é o número que indica o próximo formato a ser executado, sendo declarado somente em programa de crítica.

Um formato representa uma tela em um programa de crítica; logo, existem instruções para posicionar o cursor em uma determinada linha **(LIN)** ou coluna **(COL),** para que valores possam ser digitados **(CPO)** ou gerados **(GER/OBS).**

Podemos ainda verificar certas ocorrências antes de deixarmos que o próximo formato seja executado. Por exemplo:

```
2.1  FORM 02 (MIN 2, ALT 01);
       .
       .
       .
     outras instruções
    SEG 03:   COND  CONTROLE = 0
                    MENS ERRO6;
2.2  FORM 30;
       ESCR BRANCO (10)//CABEC1//$DATA;
       .
       .
       .
       outras instruções
     SEG;
```

No primeiro exemplo (2.1), foram utilizadas as opções de formato **MIN** e **ALT,** cuja finalidade é obrigar a que pelo menos dois campos (CPO) sejam digitados para que o formato possa ser validado e tenha como formato alternativo o de número **01.**

Caso a condição **CONTROLE= 0** seja satisfeita, o próximo formato a ser executado é o de número **03.** Do contrário, aparecerá na tela a mensagem identificada por **ERRO6** que representa uma função do tipo aritmética. Teríamos, portanto, algo como:

LTD

```
PROG EXEMPL;

FUNC ERRO6: =
"FALTA(M) CAMPO(S) A SER(EM) ATUALIZADO(S)";
```

O segundo exemplo (2.2) apresenta duas funções declaradas e uma pré-declarada. As funções pré-declaradas são iniciadas pelo caráter especial **$** e automaticamente reconhecidas pelo compilador LTD. Através delas é possível verificar ou obter:

```
1.DATA ----$VDATA  (PARAM 1)  ┐ c/pas
2.CPF  ----$CPF    (PARAM 1)  │ sagem
3.CGC  ----$CGC    (PARAM 1)  ┘ de
4.DATA ----$DATA                parâ-
5.HORA ----$HORA                metro
6.NOME
   DO
  ARQUIVO --- $TAR, etc
```

Nada impede, porém, que o usuário faça seu próprio algoritmo para calcular o dígito verificador do **CPF,** por exemplo. Logo, haverá a necessidade de declararmos a função.

Ainda no segundo exemplo, as funções declaradas **BRANCO** e **CABEC1** poderiam ter o seguinte aspecto:

```
TRANSF EXEMPL (IA=1, CTR=A, FA=7);
FUNC CABEC1:= "EXEMPLO DE FUNC. DECL.";
FUNC BRANCO (X) := $DUPL (" " ,X);
```

Observe que a função **BRANCO (10)** equivale à função pré-declarada **$DUPL (" " ,10),** ou seja, gera dez espaços.

3. FIM DE PROGRAMA

A instrução **FIM** deve ser o último comando a entrar em um programa fonte escrito em LTD.

A título de ilustração, apresentamos também alguns trechos de programas escritos em LTD.

Listagem 1 — Exemplo de programa escrito em LTD

```
0001 |*****************************************************************
0002 |                                                                 *
0003 |       ORC030  -  ESTE PROGRAMA TEM POR FINALIDADE A EMISSAO     *
0004 |                  DAS APLICACOES .                               *
0005 |                  CADA ARQUIVO CORRESPONDE A UM ORGAO.            *
0006 |                  X8 => ESTA DISPONIVEL.                         *
0007 |          DATA -  28 DE MAIO DE 1981.                            *
0008 |         AUTOR -  VALLE.                                         *
0009 |  COLABORACAO -   MELLO.                                         *
0010 |                                                                 *
0011 |*****************************************************************
0012 |
0013 TRANSF ORC030 (IA=1,CTR=A,FA=3,FT=4);
0014 |
0015 | OS CODIGOS QUE ENVOLVEM  TOTAIS CONTEM UM "T" NO CAMPO ATUALIZA.
0016 |
0017 TAB ORGAO ("IPEM - SP  ","IPEM - MG  ","IPEM - FORT","IPEM - BA  ",
0018           "DPM  - GO  ","IPEM - PR  ","IPEM - RJ  ","IPEM - PB  ",
0019           "IPEM - PE  ","INPM - SP  ","INPM - MG  ","INPM - CE  ",
0020           "INPM - MA  ","INPM - PI  ","INPM - PA  ","INPM - AM  ",
0021           "INPM - BA  ","INPM - AL  ","INPM - SE  ","INPM - GO  ",
0022           "INPM - MT  ","INPM - DF  ","INPM - RO  ","INPM - PR  ",
0023           "INPM - SC  ","INPM - RJ  ","INPM - ES  ","INPM - RS  ",
0024           "INPM - PB  ","INPM - RN  ","INPM - PE  ");
0025 FUNC A(X):=CDUPL ("=",X);
0026 FUNC B(X):=CDUPL (" ",X);
0027 FUNC T(X):=CDUPL ("-",X);
0028 FUNC CAD(X):=CCAD (X,11,01);
0029 FUNC PONTO(X):=
0030 SE X = 0
0031 ENTAO A(18)
0032 SENAO SE X <= 999
0033       ENTAO B(3)//CAD(X)[1:8]//CAD(X)[9:3]
0034       SENAO SE X <= 999999
0035             ENTAO B(2)//CAD(X)[1:5]//CAD(X)[6:3]//".."//CAD(X)[9:3]
0036             SENAO SE X <= 999999999
0037                   ENTAO B(1)//CAD(X)[1:2]//CAD(X)[3:3]//"."//CAD(X)[6:3]//
0038                         "."//CAD(X)[9:3]
0039                   SENAO CAD(X)[1:2]//"."//CAD(X)[3:3]//"."//CAD(X)[6:3]//
0040                         "."//CAD(X)[9:3];
0041 |
0042 FUNC CHOOSE(X) :=
0043 SE %[6:1] = 0
0044 ENTAO SE %[5:1] = 0
0045       ENTAO A(14)//B(3)//A(14)//B(3)//PONTO(X)
0046       SENAO A(14)//B(3)//PONTO(X)//B(3)//A(14)
0047 SENAO PONTO(X)//B(3)//A(14)//B(3)//A(14);
0048 |
0049 FUNC BRANCO(X):=
0050 SE %[6:1] = 0
0051 ENTAO SE %[5:1] = 0
0052       ENTAO SE %[4:1] = 0
0053             ENTAO  %[16:55]          | NENHUM BRANCOS
0054             SENAO  B(2)//%[16:53]    | DOIS BRANCOS
0055       SENAO B(4)//%[16:51]           | QUATRO BRANCOS
0056 SENAO B(6)//%[16:49];                | SEIS BRANCOS
0057 |
0058 *****************************************************************
0059 |
0060 FORM 1;  | INICIO DE ARQUIVO : A0,A1
0061 A01:=110;
0062 A1:="1"//%[2:6]//%[8:6];
0063 MENS "INICIO - ORC030";
0064 SEG;
0065 FORM 2;
0066 |
0067 REP A0(2:1);   | PRIMEIRA FASE : A2,A3,A4,A5,A7,A9
0068 |
0069 A2:= SE (%[3:2] = "31" E %[2:1] != "T")
0070       ENTAO %[496:11] + A2
0071       SENAO A2;
0072 A7:= SE (%[3:8] = "31110100" OU %[3:8] = "31110200")
0073       ENTAO %[496:11] + A7
0074       SENAO A7;
0075 A3:= SE (%[3:2] = "32" E %[2:1] != "T")
0076       ENTAO %[496:11] + A3
0077       SENAO A3;
0078 A4:= SE (%[3:2] = "41" E %[2:1] != "T")
0079       ENTAO %[496:11] + A4
0080       SENAO A4;
0081 A5:= SE (%[3:2] = "42" E %[2:1] != "T")
0082       ENTAO %[496:11] + A5
0083       SENAO A5;
0084 |
0085 A9:=80;        | INICIALIZA A9 (LINHAS)
0086 |
0087 FR;
0088 |
0089 REP A0(3:1);  | SEGUNDA FASE : A6,A9,?0
0090 |
0091 | A6 => PAGINA
0092 | A9 => LINHAS
0093 |
0094 ?0:= 61;
0095 A61:= SE A9 > 23
0096       ENTAO A6 + 1
0097       SENAO A6;  | ADICIONA 1 P/ PAGINA
```

Nilton do Valle Oliveira é analista de software do INMETRO — Instituto Nacional de Metrologia, Normalização e Qualidade Industrial, e faz o curso de Tecnólogo em Processamento de Dados na Faculdade Nuno Lisboa, no Rio de Janeiro.

Listagem 2 - Exemplo de programa escrito em LTD

```
0004 |*****************************************************************
0005 |                                                                 *
0006 |       AUTHOR - VALLE.                                          *
0007 |         DATE - 04/81.                                          *
0008 | COLABORACAO - MUCKEN.                                          *
0009 |*****************************************************************
0010 |
0011 |
0012 PROG LPE;
0013 |
0014       TAB % (1,2,3);
0015       FUNC TIT1:="SISTEMA DE MATERIAL";
0016       FUNC TIT2:="POSICAO DE ESTOQUE";
0017       FUNC TIT3:="CODIGO";
0018       FUNC TIT4:="ESPECIFICACAO DO MATERIAL";
0019       FUNC TIT5:="QUANTIDADE  VALOR";
0020       FUNC TIT6:="SALDO ANTERIOR";
0021       FUNC TIT7:="ENTRADA";
0022       FUNC TIT8:="SAIDA";
0023 |
0024       FUNC ERRO1:="POSICAO-1 TEM QUE SER = (1) ,(2) OU (3)";
0025       FUNC ERRO2:="CODIGO NAO PODE SER = ZEROS";
0026       FUNC ERRO3:="ESPECIFICACAO NAO PODE SER = BRANCOS";
0027       FUNC ERRO4:="CAMPO NAO PODE SER = ZEROS";
0028       FUNC ERRO5:="CAMPO DEVE SER = ZEROS";
0029       FUNC ERRO6:="FALTA(M) CAMPO(S) A SER(EM) ATUALIZADO(S)";
0030 |
0031  FORM 1 (MIN 2,ALT 2);
0032 |
0033       APG 1600;
0034       LIN 2;
0035       COL 12;
0036       OBS(24):=TIT1//"  (1)";
0037       LIN 3;
0038       COL 12;
0039       OBS(18):=TIT2;
0040       LIN 5;
0041       COL 37;
0042       OBS(6):=TIT3;
0043       LIN 7;
0044       COL 37;
0045       CPO(N11) COD1 (FC)       COND COD1 & %
0046                                MENS ERRO1;
0047            COL 38;
0048            CPO(Z14) COD2       COND COD2 != 0
0049                                MENS ERRO2;
0050 A11:=CCAD(COD1//COD2,5,80);
0051       LIN 8;
0052       COL 12;
0053       OBS(25):=TIT4;
0054       LIN 10;
0055       COL 10;
0056       GER(1):=1;
0057            COL 12;
0058            CPO(Z1A0) ESP       COND %[1:1] != " "
0059                                MENS ERRO3;
0060            COL 12;
0061            CPO(Q:60);
0062 |
0063 SEG 2;                        COND ESP != " "
```

A Schumec aumenta os recursos de seu micro M-85, acrescentando novo terminal disco rígido e prometendo para breve uma UCP de 16 bits, em multiprocessamento.

Schumec faz mudanças no seu M-85

O M-85 da Schumec está de cara nova, hardware ampliado e com a aprovação da SEI dando-lhe um novo status. A empresa resolveu deixar um pouco de lado o usuário pessoal e hobbista e partir firme para o mercado dos profissionais, empresas e controle de processos, dotando o M-85 de um novo terminal de vídeo, disco rígido e prometendo um microprocessador de 16 bits, ainda este ano, para trabalhar com multiprocessamento.

CUSTO ALTO PARA HOBBISTAS

Há pouco tempo atrás a Schumec oferecia o M-85 em duas configurações: cassete e disquete, ambos com um televisor comercial transformado em terminal de vídeo (veja a matéria "Micro-85, da Schumec", MS nº 8, maio/82). Na versão cassete ele trazia a vantagem de utilizar um gravador stereo Gradiente, do tipo tape-deck, que procurava garantir uma gravação de dados e programas com muito mais segurança do que os cassetes normalmente utilizados. Esta versão voltava-se ao usuário pessoal e hobbista, oferecendo qualidade na hora de armazenamento de programas.

Entretanto, após algum tempo com o M-85 no mercado, a Schumec chegou à conclusão que esta versão acabava por ficar um pouco cara para um usuário que não era muito exigente. "Na realidade", comenta o Eng. Carlos Roussenq, um dos diretores da Schumec, "um usuário pessoal não faz tanta questão de uma gravação um pouco melhor. Se der um erro na leitura, ele vai e faz uma nova leitura. Quando ele é realmente exigente, ele passa logo para o disquete. Muitos dos nossos primeiros clientes logo quiseram comprar o disquete quando sentiram o potencial do M-85".

Sensível a isto, a Schumec tirou essa versão cassete de linha, embora continue a dar manutenção normal para os que a possuam. Além disso, resolveu concentrar forças no mercado profissional e fez importantes acréscimos ao M-85.

Uma das primeiras coisas que se pode notar no novo M-85 é seu terminal de vídeo. Desenvolvido pela própria Schumec, ele já conta com a aprovação da SEI e está sendo comercializado em esquema OEM. Ele vem com tela anti-reflexiva, display de 25x80 com letras minúsculas e maiúsculas, vídeo-reverso, campo piscante, cursor endereçável por software e modo de operação semi-gráfico com 72x 160 pontos. Um teclado ASCII acompanha o terminal, com ou sem teclado numérico reduzido.

"A principal vantagem do nosso terminal", diz o Eng. Carlos, "é que ele contém todas as qualidades de um Saggita (terminal fabricado pela Scopus), além de operar em modo gráfico, e custa praticamente a metade do preço. É um excelente terminal, e por um baixo custo".

A nova imagem do Schumec, na foto com um móvel opcional que acondiciona a UCP, uma unidade de disquete e outra de disco rígido.

A outra novidade são os discos rígidos. O M-85 pode utilizar até quatro discos rígidos da Multidigit, tecnologia Winchester, de 6 ou de 12 megabytes, o que amplia sua capacidade de armazenamento para até 48 Mb. Os discos rígidos podem ainda ser utilizados junto com disquetes de 8" (face e densidade simples) o que amplia sua flexibilidade.

16 BITS MODULAR

Mas não param aí as modificações no M-85. Graças à utilização do barramento S-100, diversas placas adicionais estão nos planos da Schumec para muito breve.

A mais importante delas é a da UCP 8088, da Intel, de 16 bits, que permitirá ao M-85 trabalhar em multiprocessamento, com até quatro terminais e com o sistema operacional MP/M. Para quem possuir um M-85 com a UCP 8085, a mudança de processador será simples e barata, bastando, praticamente, trocar o chip.

Do mesmo modo, aproveitando sua configuração modular, o M-85 terá disponível um formatador de disquetes para 5 1/4" e para 8" face e densidade duplas, plaquetas com memória RAM que possibilitarão sua expansão para até 256 Kb e a impressora Emília, da Elebra, que será comercializada junto com o micro.

Tudo isto dotará o M-85 de uma configuração modular com uma flexibilidade que pode atender tanto a profissionais liberais como a empresas de maior porte, tudo a partir de uma mesma estrutura básica, atendendo ainda à área de controle de processos, onde o M-85 já atua em empresas como a CEDAE e a Souza Cruz, com software e hardware adicional implementados pela Schumec.

A Schumec Sistemas Ltda. fica na Rua Barata Ribeiro 370/loja 305, Copacabana, Rio de Janeiro, e na Av. Paulista 807/sala 1617, em São Paulo.

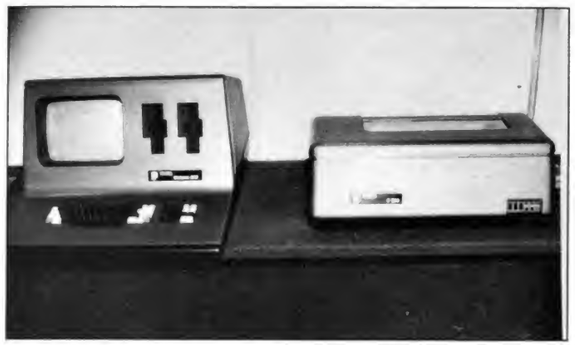
Texto: Paulo Henrique de Noronha
Foto: Monica Leme

Microfestival 83

Na primeira semana de março o público paulista teve oportunidade de assistir ao **Microfestival 83, Primeiro Encontro Brasileiro de Microinformática,** patrocinado pela Guazelli Associados, Revista MicroMundo e a Compucenter, que ocupou por quatro dias o Palácio de Convenções do Anhembi, com cerca de 50 stands de fabricantes e diversas palestras sobre microcomputadores e suas aplicações.
Vários lançamentos movimentaram a mostra, que, na opinião da maioria dos expositores, foi visitada por um público (cerca de 10 mil pessoas em todos os dias) realmente interessado em conhecer os equipamentos com vistas a uma futura utilização e não apenas por mera curiosidade.
Veja nestas duas páginas os principais lançamentos de micros, periféricos e software que aconteceram durante o **Microfestival 83.**

No stand da Computerland foi lançado o TRS-80 Modelo IV, fabricado pela SAYFI Computadores Ltda., de São Paulo, que adquiriu os direitos de utilização no Brasil dos nomes Radio Shack e TRS-80. À venda com exclusividade na Computerland, o TRS-80 Modelo IV é um aperfeiçoamento do Modelo III da Radio Shack americana, com a incorporação de alguns comandos do Modelo II, o que propicia a utilização de cores. Em sua configuração básica, o TRS-80 Mod. IV está custando cerca de Cr$ 750 mil.

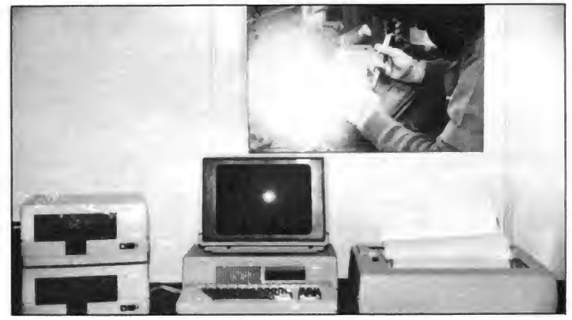
Uma das novidades que mais chamou a atenção dos visitantes foi micro EGO, fabricado pela Softec, um poderoso computador multiusuário de 16 bits, compatível com o Personal Computer da IBM americana. Seu microprocessador é o 8088 e ele trabalha com sistema operacional CP/M86 ou Analix (uma versão do UNIX), comportando até sete terminais, 256 Kb de RAM, vídeo gráfico colorido e discos rígidos de 5 e 10 Mb. Seu preço, em configuração com dois disquetes e 64 Kb RAM, é cerca de Cr$ 5 milhões.



A Prológica lançou o Sistema 600, micro com o visual e o hardware do Sistema 700, porém utilizando disquetes de 5 1/4", que pretende ocupar a faixa intermediária entre o CP-500 e o Sistema 700, com preço inicial de Cr$ 2,5 milhões. Para o CP-500, a Prológica mostrou a impressora serial P-500, de 80 colunas e velocidade de 80 CPS, que será comercializada a partir de maio, por volta de Cr$ 400 mil.

Outros lançamentos

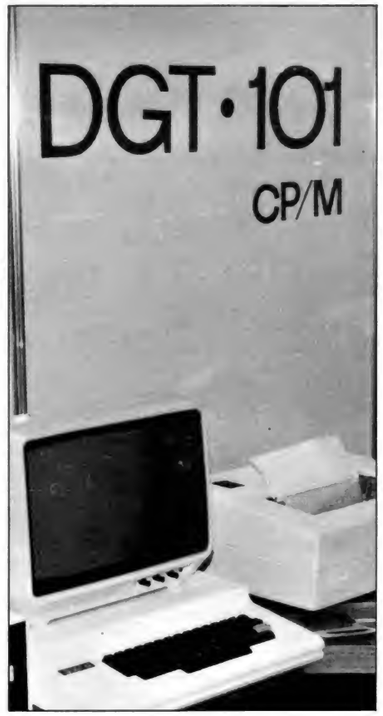
Approach – Sistema de relatórios e classificação de arquivos INFOSTAR. Comercialização no segundo semestre de 83.
Cobra – Disco rígido de 10 Mb para o Cobra 305.
Dataroad – Software para micros compatíveis com Apple.
Elebra – Impressoras Emília II, com caracteres gráficos, e Mônica, bidirecional com velocidade de 80 CPS e preço aproximado de Cr$ 600 mil.
Hengesystems – Fancy, pacote de hard/software para micros compatíveis com Apple, para uso de pessoas leigas em computação, vindo em duas configurações: mínima, com circuito e dois disquetes, cerca de Cr$ 400 mil; e máxima, com circuito e cinco disquetes, aproximadamente Cr$ 500 mil, base ORTN/março.
LABO – Labo 8221 com 256 Kb RAM, saída para três terminais e discos rígidos de 12 Mb. Comercialização no segundo semestre de 83.
LPRINT – Interface para uso de máquina de escrever elétrica IBM como impressora por microcomputadores. Preço aproximado de Cr$ 350 mil.
Memphis – Compartimento Magnitex para guardar disquetes e outros suprimentos.
Microdigital – TK85 com 10 Kb ROM, 16 Kb RAM, teclado tipo calculadora, funções VERIFY e HIGH-SPEED, saídas para TV, cassete e impressora. Preço Cr$ 180 mil, configuração mínima.
Microtec – Módulo de expansão de 16 Kb RAM e interface MT 3 AD para E/S de dados de fontes analógicas, ambos para o MT 300.
Scopus – Software para o MicroScopus nas áreas administrativas e comercial e Cálculo do FGTS.
Setra – Interface para uso de máquinas de escrever IBM esfera como impressoras para microcomputadores. Preço aproximado Cr$ 450 mil.
Sharp – Computador de bolso PC 1211, por Cr$ 150 mil em versão simples e Cr$ 217 mil com impressora.
Spectrum – Expansão de 16 Kb RAM e diversos programas para o Microengenho.
Unitron – Expansão de 32 Kb RAM e interface para TV à cores para o AP II.
VM Consultoria – Sistemas aplicativos para micros pessoais.

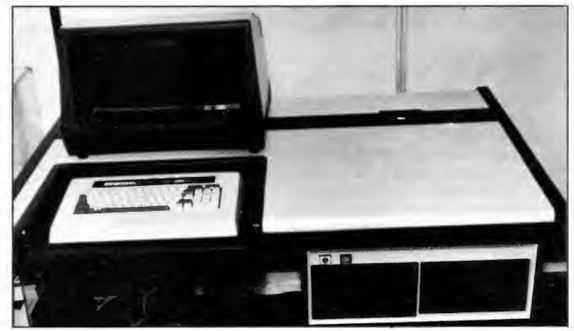


O I-7000 (ex-I-7010) da Itautec teve seu lançamento oficial durante o Microfestival, com algumas modificações no seu modelo original. Ele agora está com microprocessador NSC 800 da National, ao invés do 8085, e em termos de software a Itautec está oferecendo um Sistema Emulador de Terminal e um processador de textos de nome **Redator**. *O I-7000 com UCP, teclado e vídeo está custando cerca de Cr$ 1,5 milhão, preço de março.*

Compatível em software com TRS-80 modelos I e III, o micro JR atraiu grande número de visitantes ao stand da Sysdata. Em sua configuração mínima, com 16 Kb de RAM, o JR Sysdata está com preço de lançamento em torno de Cr$ 320 mil.

A Digitus aproveitou o Microfestival para lançar o DGT-101, uma versão do DGT-100 que comporta o sistema operacional CP/M. Além disso, o DGT-100 conta agora com monitor de fósforo verde, placas de expansão para 64 Kb RAM, placa para uso do CP/M e interface para uso de TV a cores. Em termos de software, foram lançados um Sistema de Controle de Estoque, um Processador de Texto e um Editor de Assembler.





A Brascom apresentou seu micro **BR 1000** *em versão multiusuário, com dois terminais e disco rígido Winchester de 5 Mb. Além disso, chamou atenção o lançamento de uma impressora margarida, a* **Daisy Writer,** *que trabalha com cartucho de fita do tipo usado em máquinas de escrever IBM e tem preço aproximado de Cr$ 2 milhões.*

Cobertura: Stela Lachtermacher e Beatriz Carolina Gonçalves
Fotos: Carlão Limeira

classificados

troco · financio · ofereço · vendo · alugo · compro

• Engenheiro júnior em desenvolvimento de produto na área digital, hard e software, procura colocação em empresa de médio ou grande porte, cuja finalidade seja a informática. Maiores informações com Itamar, tels.: (011) 544-2594 e 433-4623, SP.

• Troco programas para os micros ZX-81, TK82-C, NE-Z8000 e CP-200. Escrevam para Renato Strauss, Rua Cardoso de Almeida, 654/32, CEP 05013, SP.

• Compro os números 6 e 7 de MICRO SISTEMAS. Paulo Túlio Radicchi, Rua Costa Rica, 66, Bairro Sion, CEP 30000, tels: (031)225-7761 ou 221-2778, Belo Horizonte, MG.

• Vendo números atrasados das revistas BYTE, INTERFACE AGE, COMPUTER GRAPHICS, SIGSMALL, SIGMICRO e SIGDOC (as quatro últimas publicadas pela ACM). Vendo também os seguintes livros publicados pela ACM: "Proceedings of the 3rd symposium on small systems", setembro de 80, Cr$ 5 mil; "Proceedings of the 2nd symposium on small systems", outubro de 79, Cr$ 5 mil; "13th microprogramming workshop", nov/dez de 80, Cr$ 5 mil; "Conference proceedings in computer graphics", agosto de 81, Cr$ 8 mil. Os interessados devem procurar Gerson, Rua Eça de Queiroz, 288, apt. 24, CEP 04011, tel.: (011) 549-6870, SP.

• Vendo programas TRS-80 modelos I e III e CP-500: xadrez, bridge e outros. Cartas para Giselle Camargo, Caixa Postal 8438, Curitiba, PR.

• Compro TK82-C com 16 Kb RAM ou CP-200. Se você tem um destes equipamentos e quer vendê-lo, podemos realizar um negócio vantajoso para ambos à base de troca e/ou dinheiro. Tratar com Eduardo Brunstein pelo tel.: (011)67-6662, SP.

• Dou aula de BASIC em minha casa aos sábados e tenho muitos programas em BASIC e linguagem de máquina. Vendo ou troco. José Carlos Taveira, tel.: (031)441-5155, Belo Horizonte, MG.

• Vendo microcomputador Casio de bolso FX-702: linguagem BASIC, alfanumérico, memória constante, impressora e interface para cassete. Tratar com Rubens pelo tel: (011) 453-7328 ou 455-4962, SP.

• Vendo jogos para o TK82-C, como Simulador de Vôo, Labirinto e Sicom. Preços abaixo da tabela. Tratar com Rony, tel.: (011)287-3633, SP.

• Compro os exemplares números 6 e 7 de MICRO SISTEMAS, desde que em bom estado. Marcílio, tel.: (041) 224-2647, Curitiba, PR.

• Disponho de vários programas para os micros ZX-81 e TK82-C. Aos interessados em trocá-los ou comprá-los, entrar em contato com Marcelo Rodrigues, Rua J. Carlos, 90/701, CEP 22461, tel.: (021) 286-4765, RJ.

• Vendo HP-41CV, sem uso, na caixa, por $ 290. Tratar com Alexandre, Rua das Azaléas, 32, Mirandópolis, CEP 04049, tel: (0187)275-5073, SP.

Clubes

• Sou proprietário de um TRS-80 Model II e estou interessado em fazer intercâmbio de disquetes de COBOL, jogos etc. José Carlos Jorge, tel.: (021)273-5258, RJ.

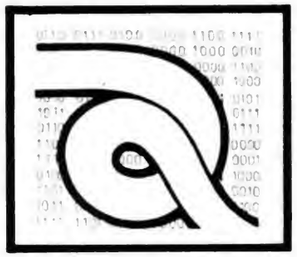
• Sou engenheiro civil, trabalho na CHESF — Companhia Hidroelétrica do São Francisco e tenho interesse na permuta de programas de qualquer área. Antonio Francisco Militão Rufino, Av. Luiz de Lacerda, 277, apt. 202, Iputinga, CEP 50000, Recife, PE.

• Desejo entrar em contato com pessoas que possuam um DGT-100 para troca de idéias e programas. Gostaria também de obter informações sobre a linguagem de máquina aplicada a este micro, assim como dados sobre os jogos Senha, Galáctica e Simulador de Vôo. Escreva para Gerson Lidak, Rua Eduardo Coutre, 133, Jardim Santa Bárbara, CEP 80000, Curitiba, PR.

• Gostaria de formar um grupo de usuários de micros pessoais em Belo Horizonte. Interessados (ou grupos já formados, se houver), favor contactar José Ribeiro Pena, Rua Trifana, 529/101, CEP 30000, tel.: (031)223-7860, Belo Horizonte, MG.

• Gostaria de me corresponder com usuários de microcomputadores. Trabalho com • S-700, CP-500, CP-200 e NE-Z8000. Alexandre Borrego, Conj. Residencial Massari, 131, CEP 12300, tel.: (0123) 51-2508, Jacareí, SP.

• Para troca de idéias e programas, enviar cartas para Clifford Alves Miller e Fernando Martins Bauer, Rua Guilherme Lahm, 555, Cx. Postal 49, CEP 95600, Taquara, RS.



Curso de Assembler — III

O microprocessador Z80 foi adotado como processador base para este curso pelas seguintes razões:

a) ele é largamente utilizado em microcomputadores no Brasil;
b) possui um repertório de instruções bastante extenso, possibilitando o desenvolvimento de programas mais rápidos e que ocupam menos memória;
c) possui um conjunto de registradores que facilitam a programação em linguagem de máquina;
d) é totalmente compatível com as instruções de máquina do microprocessador 8080 da Intel, que é o mais utilizado em todo o mundo, embora conte com um conjunto de registradores e instruções bem inferior ao Z80;
e) encontra-se com facilidade literatura e montadores Assembler para ele.

Justificada a razão da escolha do microprocessador Z80 como base para este curso, vamos dar início à nossa terceira lição.

REGISTRADORES

O microprocessador Z80 tem dois conjuntos de oito registradores de 8 bits de propósito geral, quatro registradores de 16 bits e dois registradores especiais de 8 bits.

Um registrador é uma localização de memória interna no microprocessador, que é usada durante o processamento de uma instrução. Um dos conjuntos de registradores de propósito geral de 8 bits é chamado de conjunto de registradores **principais,** enquanto o outro é o conjunto de registradores **alternativos.** O conjunto principal é usado durante a execução de uma instrução e o conjunto alternativo somente é usado por duas instruções, que trocam o conteúdo do conjunto principal com o conjunto alternativo.

Os registradores de propósito geral são chamados pelos nomes **A, F, B, C, D, E, H** e **L.** O registrador **A** também é chamado **Acumulador** e é o mais importante dos registradores, pois é nele que ocorrem a maior parte das ações do microprocessador.

O registrador **F** também é chamado de **Flag,** porque seus bits indicam várias condições internas e ele nunca é usado diretamente por uma instrução: seus bits são automaticamente posicionados de acordo com o resultado da execução de uma instrução.

Os registradores restantes **B, C, D, E, H** e **L** podem ser usados como registradores de 8 bits ou, em pares, como registradores de 16 bits. Neste caso, **B** e **C, D** e **E, H** e **L** são usados aos pares, chamados de **BC, DE** e **HL.**

Na figura 1 podemos ver um diagrama que nos mostra a estrutura interna do microprocessador Z80.

Dois registradores de 16 bits são chamados registradores **indexadores,** designados por **IX** e **IY.** Eles são usados como ponteiros de posições de memória e um valor é somado ao seu conteúdo para determinar o local de memória desejado.

Os outros dois registradores de 16 bits são chamados de **Stack Pointer (SP)** e **Program Counter (PC).** Este último determina a ordem de execução das instruções. Quando é iniciada a execução de uma das instruções, o **PC** contém o endereço da próxima instrução que será executada. Uma instrução de desvio modifica o conteúdo do **PC.**

O **Stack Pointer** contém um endereço que aponta para a área livre de memória, que é usada para armazenamento temporário de dados durante a execução de um processamento. Se a área que ele aponta for destruída, ou se apontar para uma área de memória não existente, um desastre pode ocorrer!

Os registradores restantes de 8 bits são chamados de registrador de **Interrupção (I)** e registrador de **Refresh (R).** O registrador de **Refresh** torna fácil e simples a utilização de memórias dinâmicas. Entretanto, este registrador está voltado a auxiliar o hardware existente no microcomputador, sendo totalmente transparente para o programador

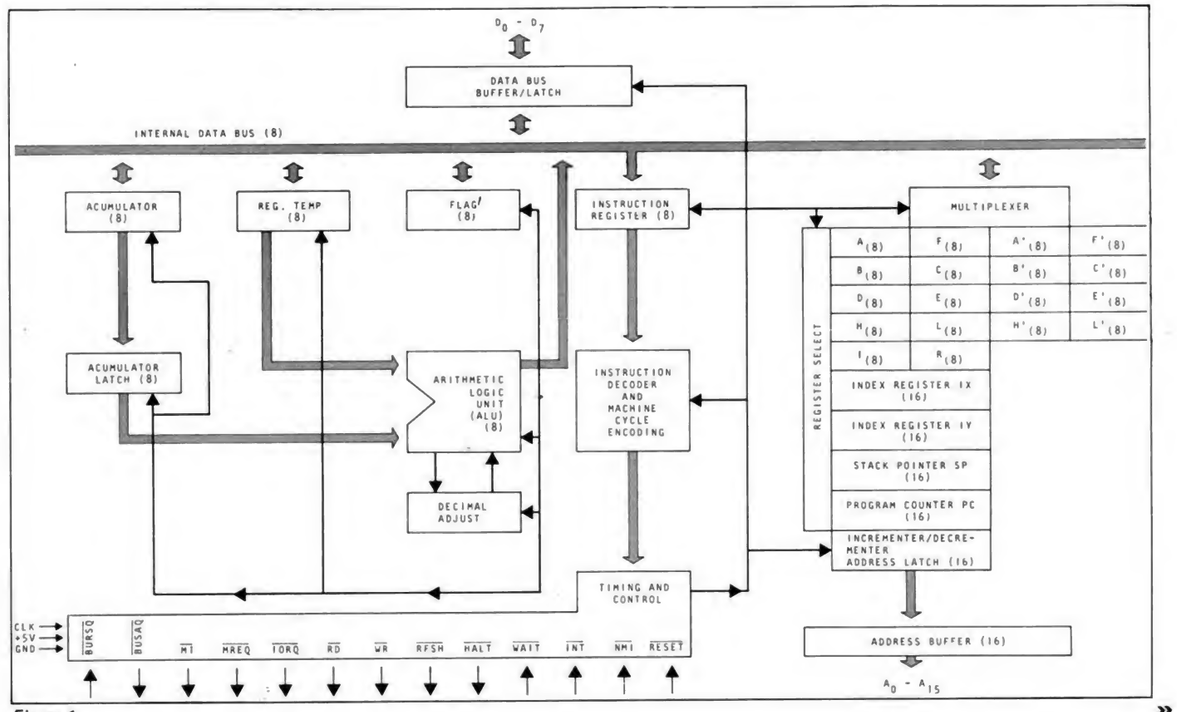


Figura 1

O registrador de **Interrupção** possibilita uma maior flexibilidade no sistema de interrupções do Z80. As interrupções são usadas somente em sistemas em tempo real, que envolvem uma avançada programação que foge do escopo deste curso.

Z80 x 8080

Normalmente se deseja conhecer as diferenças entre os microprocessadores Z80 e 8080. O 8080 possui os mesmos oito registradores de propósito geral de 8 bits que o Z80 tem, mas não possui o conjunto de registradores alternativos, os indexadores e nem os registradores de **Interrupção** e **Refresh.** O conjunto de instruções do Z80 é muito mais extenso que o do 8080 porque inclui ainda todas as instruções que manipulam estes registradores.

Em resumo, estas diferenças facilitam bastante a vida do programador quando ele está usando um microcomputador baseado no Z80. O conteúdo de algum par de registradores pode ser facilmente salvo e recuperado utilizando-se as facilidades da pilha de dados **(Stack).** Um conjunto de registradores pode ser trocado com o conjunto alternativo e, além disso, o Z80 dispõe de várias macro-instruções, como, por exemplo, a instrução **LDIR,** que move um bloco de memória.

A programação em Assembler num micro com recursos equivalentes ao Z80 torna-se muito mais fácil e segura, principalmente quando se está iniciando na programação em linguagem de máquina.

O USO DE REGISTRADORES

Há certas operações que só podem ocorrer em determinados registradores. O registrador **A,** o **Acumulador**, é o mais importante registrador, conforme já dissemos. Todas as operações de aritmética e lógica de 8 bits envolvem o **Acumulador** (que contém um dos operadores) e o resultado da operação sempre é colocado nele. Além disso, ele também é utilizado por algumas instruções que realizam leitura ou gravação de bytes na memória. O registrador **Flag (F)** é a outra metade do registrador **A**.

Assim como os registradores **A** e **F** são agrupados pelo microprocessador, todos os registradores podem ser tratados em grupos de dois bytes. O par de registradores **HL,** por exemplo, tem dois usos primários. Primeiramente ele é o "acumulador" para operações aritméticas de 16 bits (não há operações lógicas com 16 bits). Todas as operações aritméticas de 16 bits usam o registrador **HL** para armazenar um dos operandos e o resultado também é colocado nele pelo microprocessador. Além disso, ele pode ser usado como ponteiro do endereço de memória que deseja ler ou gravar um byte. Esta operação é normalmente utilizada e o operando é indicado pelo **HL.** Algumas vezes os registradores **BC** e **DE** podem ser usados da mesma maneira (no código mnemônico do 8080 o operando **HL** é especificado por **M**, que significa **Memória**).

O registrador individual **B** e o par de registradores **BC** são usados como contadores do número de vezes que um conjunto de instruções deve ser repetido. O registrador **B**, com a instrução **DJNZ**, é usado como um contador cujo mnemônico significa: **Decremente B e desvie para a localização especificada se o contador for diferente de zero.** O par de registradores **BC** é usado como um contador para todas as instruções de transferência de bloco (**LDI, LDIR** etc). Estas instruções são usadas para mover um bloco de bytes de uma localização de memória para outra. O registrador **C** somente é utilizado para certas operações de entrada e saída.

O par de registradores **DE** é usado de maneira análoga a **HL** e **BC**. Os registradores **BC** e **DE** podem ser utilizados para especificar um dado endereço, mas só movimentam o acumulador com a memória.

FLAGS

O registrador **Flag (F)** nunca é utilizado para manipular dados. Ele contém bits lógicos chamados **flags,** que são setados ou ressetados de acordo com o resultado das operações. O **F** é um registrador de 8 bits, embora apenas 6 bits representem **flags** e, destes, apenas 4 sejam realmente importantes para a maior parte de seus programas.

Estes quatro flags são chamados de **Zero (Z), Flag de Sinal (S), Carry Flag (C)** e **Flag de Paridade/Overflow (P/V).** Os outros dois **flags,** o **Carry Parcial (H)** e o **Flag de Soma/Subtração (N),** são usados somente com a instrução **DAA** (Ajuste Decimal do Acumulador), que somente é utilizada para operações com números **BCD.**

O **Carry Flag** é setado normalmente quando uma instrução produz um resultado que é um bit maior que o valor que pode estar contido em um registrador simples. Da mesma forma, ele também é setado quando uma operação de subtração produz um empréstimo (**borrow**). O **Carry Flag** é ainda afetado por instruções de **Shift** e **Rotação** e é ressetado (zerado) por operações lógicas. **No Carry** é indicado por **NC.**

O **Zero Flag** é setado somente se o resultado é zero. **Não Zero** é indicado por **NZ.** O **Flag de Sinal** indica as condições **Mais (P)** ou **Menos (M)** e é uma cópia do bit de sinal (bit 7) do **Acumulador.** Os três **flags, Zero, Sinal** e **Carry** podem também ser setados por instruções de comparação.

O **flag P/V** indica uma condição de paridade par (**PE**) ou ímpar (**PO**), isto é, se o número de bits setados em um byte formam um número par ou ímpar. Ele é usado para testar condições de estouro **(overflow)** e para indicar paridade, dependendo da instrução. O **P/V** também é utilizado para outros fins, como por exemplo durante a execução de uma instrução de transferência de um bloco de memória.

As operações de comparação são equivalentes à subtração, mas com uma importante diferença: os valores contidos nos registradores não são afetados. As instruções de comparação são seguidas por instruções de desvio condicional **JP** (Desvio Absoluto) ou **JR** (Desvio Relativo). Podemos ver as instruções assumidas em uma comparação na figura 2.

FLAG	RESULTADO DO TESTE
Z	O valor comparado é igual ao conteúdo do acumulador.
NZ	Os dois valores são diferentes.
C	O conteúdo do acumulador é menor que o valor comparado.
NC	O conteúdo do acumulador é maior ou igual ao valor comparado.
M	O sinal do conteúdo do acumulador é menor que o valor comparado.
P	O sinal do conteúdo do acumulador é maior que o valor comparado.
PO	Um overflow foi produzido durante a operação de comparação.
PE	Não houve overflow durante a operação de comparação.

Figura 2

MODOS DE ENDEREÇAMENTO

Para executar alguma operação envolvendo memória, o computador precisa conhecer os endereços da localização de memória envolvida. Para conveniência de programação, há várias maneiras de se realizar o endereçamento. Os modos de endereçamento possíveis no Z80 são os seguintes:

a) Imediato – Um byte contido na instrução é movido para o registrador. Exemplo:
LD A,1

O valor **1** é carregado no registrador **A.**

b) Imediato extendido – Neste caso, dois bytes contidos na instrução são carregados em um par de registradores. Exemplo:
LD HL,1000

O par de registradores **HL** é carregado com o valor **1000.**

c) Relativo – Aplicado somente a instruções de desvios relativos **(JR).** O valor no byte seguinte à instrução é somado ao conteúdo do **program counter** para determinar o novo endereço. O endereço indicado deve estar no intervalo -128 a +128 bytes. Exemplo:
JR $+10
onde **$** significa o endereço corrente da instrução. Neste caso, é provocado um desvio para 10 bytes seguintes ao endereço corrente.

d) Extendido – O endereço do operando é especificado na instrução. Exemplo:
LD A,(1000)

O registrador **A** é carregado com o conteúdo do endereço **1000** de memória.

e) Indexado – O endereço de um operando é determinado pela soma de um byte chamado **deslocamento** ao valor contido no registrador indexador. Exemplo:
LD A,(IX+5)

O registrador **A** é carregado com o conteúdo de memória apontado pela soma de **5** no registrador **Indexador (IX).**

f) Registrador – O conteúdo de um registrador é carregado para outro. Exemplo:
LD B,C

O conteúdo do registrador **C** é carregado no registrador **B.**

g) Implícito – Neste modo, um registrador não é declarado na instrução, porém é automático o seu envolvimento. Exemplo:
SUB B

O registrador é subtraído do registrador **A** (que está implícito na instrução).

h) Registrador Indireto – O endereço de um operando está contido em um par de registradores (**BC, DE** ou **HL).** Exemplo:
LD A,(BC)

O conteúdo da localização de memória apontada pelo par de registradores **BC** é carregado no registrador **A.**

i) BIT – Um bit de um registrador é setado, ressetado ou testado. Exemplo:
SET 6,B

O bit **6** do registrador **B** é setado.

j) Página Zero Modificada – Aplicado somente a instruções de **Restart (RST).** O endereço deve ser um múltiplo de **8,** entre 0 e 56. Exemplo:

RST 8

Um desvio é provocado para o endereço **8,** retornando em seguida ao local de chamada.

STACK

O **Stack** é uma área de memória onde os valores contidos nos registradores podem ser salvos e recuperados. O **Stack Pointer (SP)** é um registrador de 16 bits que contém o endereço corrente do topo da área de **Stack.** A necessidade de uma área de **Stack** pode parecer estranha, visto que os valores contidos nos registradores podem ser salvos e recuperados por instruções de **LOAD(LD).**

Esta idéia de se ter uma área geral na memória para salvar e recuperar dados é muito boa, porque esta necessidade ocorre frequentemente durante a execução de um programa. O **Stack** não reside em nenhuma área particular da memória, sendo sua localização determinada pelo programador, utilizando-se de instruções que manipulam o **Stack Pointer.**

O **Stack** é organizado como um sistema **Last In-First Out (LIFO),** isto é, o último dado a entrar na pilha de dados **(Stack)** é o primeiro a sair. Na figura 3 vemos demonstrada a estrutura de uma pilha de dados.

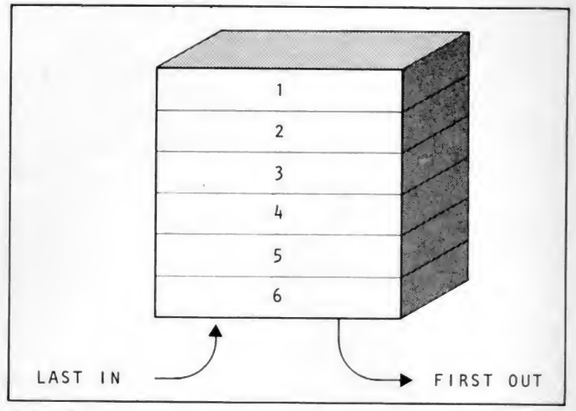


Figura 3

Quando novos dados são introduzidos no **Stack,** eles são salvos nesta área de memória e o **Stack Pointer** é decrementado de 2. Quando este dado é recuperado do **Stack,** o **Stack Pointer** é incrementado de 2.

O **Stack** é usado somente para registradores de 16 bits. A movimentação da pilha de dados **(Stack)** é feita através das instruções **PUSH** e **POP.** A instrução **PUSH** salva o conteúdo de um registrador de 16 bits no **Stack. POP** recupera os dados no **Stack.**

Você pode mover para o **Stack** os registradores **AF, BC, DE, HL, IX** e **IY.** Por exemplo, se o conteúdo do **Stack Pointer** é **4288H,** após a execução da instrução **PUSH HL** o microcomputador salva o conteúdo do registrador **H** na locação **4287H,** o de **L** em **4286H** e altera o conteúdo do **Stack Pointer** para **4286H.**

Outro uso do **Stack** é através das instruções **CALL** e **RETURN.** Estas instruções são utilizadas para desviar o fluxo de processamento para uma sub-rotina (através do **CALL**) e para retornar de uma sub-rotina para o ponto de chamada (através de **RET**).

Uma sub-rotina é uma porção de um programa, que pode ser chamada a partir de diversos pontos do mesmo e que, ao terminar, retorna à locação de memória imediatamente seguinte ao ponto de chamada. Toda vez que for executado um comando **CALL**, o endereço é salvo no **Stack** e ao ser processado o **RET** pelo microcomputador, é visto no **Stack** o endereço a partir do qual o microprocessador deve continuar a execução do programa.

Na próxima aula, daremos uma olhada no Set de Instruções do Z80. Até lá.

Amaury Correa de Almeida Moraes Junior é formado pelo curso de Análise de Sistemas da FASP, tendo feito diversos cursos de aperfeiçoamento nas áreas de eletrônica digital e microprocessadores.
Amaury trabalha como Analista na PRODESP, na área de mini/microcomputadores e presta consultoria a empresas para a implantação de sistemas de microcomputadores.

76

cursos

• O CEDM — Cursos de Aperfeiçoamento Técnico — está promovendo seu "Curso de Eletrônica Digital e Microprocessadores" por correspondência. Dividido em 36 grupos, o curso oferece material de prática (kits) e, no final, um "kit surpresa" como "prêmio de formatura". O aluno receberá um certificado de conclusão do curso. O endereço para o pedido é Caixa Postal 1642 CEP 86100 — Londrina, PR. O endereço do CEDM é R. Piauí, 191, sls. 31 e 34. Tel.: (0432)23-9674.

• A Minas Digital ministra cursos de Digitação e Programação BASIC. O curso de Digitação pode ser iniciado a qualquer hora, sendo um aluno para cada máquina. Custa três parcelas de Cr$ 6 mil e 200 ou à vista com 10% de desconto. O curso de BASIC custa três parcelas de Cr$ 11 mil ou Cr$ 30 mil à vista. Podem ser ministrados cursos de BASIC, fechados, para firmas. A Minas Digital fica na R. Tupinambás, 1045, cjs. 601/602. O telefone é (031)201-7555, Belo Horizonte, MG.

• A SDI está ministrando cursos semanais de BASIC, com metodologia inédita no Brasil, baseada no similar americano. O curso vai permitir ao aluno fazer seu curso de BASIC em casa. Maiores informações na própria SDI, Av. Brig. Faria Lima, 1853, cjs. 511/512, tel.: (011)813-4031, São Paulo, SP.

• A Fundação de Desenvolvimento da Pesquisa, FUNDEP, através da Universidade Federal de Minas Gerais, UFMG, segue com seu Programa de Cursos para 1983: "Programação de Computadores para Aplicações Técnico-Científicas", de 02/05 a 25/08, ao preço de 65 ORTNs; "Análise de Sistemas de Informação", de 09/05 a 09/12, ao preço de 163 ORTNs; "Seminário: Metodologia para o Desenvolvimento de Sistemas", de 13/05 a 16/06, ao preço de 35 ORTNs; "Eletrônica Digital", de 02 a 14/05, ao preço de Cr$ 67 mil. Informações podem ser obtidas pelo tel.: (031)441-8077, R. 1447 e 1451. A FUNDEP fica na Av. Antônio Carlos, 6627 — Pampulha, Belo Horizonte, MG.

• A ACI — Assessoria e Controles Internos S/C Ltda. divulga sua Programação de Seminários e Conferências. Para maio, a seguinte conferência: "A Metodolgoia "Fato" para a Realização Eficiente da Auditoria de Sistemas Computarizados", nos dias 06/05 (em Curitiba), 03/05 (em São Paulo) e 04 /05 (no Rio). Maiores informações pelo tel.: (011)280-5648 ou na R. Tabapuã, 627 — 6º andar, cj. 62, São Paulo, SP.

• A MICROMAQ realizará dois cursos em abril/maio de 1983: "Circuitos Combinacionais", de 25 a 29/4, das 18:30 às 21:30h, ao preço de Cr$ 30 mil; "Circuitos Sequenciais", de 2 a 6/5, das 18:30 às 21:30h, ao preço de Cr$ 30 mil. Informações e reservas na MICROMAQ, R. Sete de Setembro, 92, lj. 106, Centro, Rio de Janeiro, RJ. Tel. é (021) 222-6088.

• O Management Center do Brasil promove um Curso de Programação BASIC com o apoio de microcomputadores. O preço do curso é Cr$ 99 mil para sócios e Cr$ 116 mil 500 para não-sócios. O MCB fica na Av. Paulista, 1765, 11º andar, tel.: (011)284-8211.

• A NASAJON Sistemas está oferecendo um curso de Microcomputadores, ministrado durante duas semanas e com turmas de 10 alunos. O curso custa Cr$ 25 mil e será realizado de segunda a sexta das 19:00 às 21:00h. Durante o curso, serão distribuídas apostilas para acompanhamento das aulas e também manuais da linguagem BASIC. A NASAJON Sistemas fica na Av. Rio Branco, 45, grupo 1311, tel.: (021) 263-1241, Rio de Janeiro, RJ.

• A SAD — Sistemas de Apoio à Decisão — divulga sua Programação para 1983. Serão os seguintes cursos/seminários: "O Visicalc", dias 28 e 29/04, das 08:30 às 17:30h, ao preço de 20 ORTNs; "A Linguagem Pascal", dias 5 e 6/05, das 08:30 às 17:30h, ao preço de 20 ORTNs; "O Sistema Operacional CP/M", dias 14 e 15/04, das 08:30 às 17:30h, ao preço de 20 ORTNs; "Redes e Teleprocessamento com Micros", dia 10/05, das 08:30 às 17:30h, ao preço de 12 ORTNs. Todos os cursos ou seminários estão limitados a 20 participantes com direito a material didático e um micro para cada dois participantes. Serão realizados na sede da SAD, que fica na R. Cardoso de Almeida, 993, tel.: (011)864-7799, São Paulo, SP.

• A FUPAI — Fundação de Pesquisa e Assessoramento à Indústria vai promover, de 9 a 14 de maio, um curso sobre "Microprocessadores 8085 — Introdução e Aplicações" com duração de 60 horas e taxa de inscrição de Cr$ 85 mil 800. Inscrições e informações na FUPAI, Rua Cel. Rennó, 7, Itajubá, Minas Gerais. Tel.: (035)622-3477

• Dando prosseguimento à sua programação, a J. HEGER, estará promovendo, de 9 a 20 de maio, o curso "Programação para HP-41C/CV", das 19 às 22:30hs ao preço de Cr$ 30 mil. O endereço da J. HEGER é Av. Moaci, 155, Moema, São Paulo. Tels.: (011) 531-7158 e 531-7324.

• A MICROSHOP promove regularmente, em semanas intercaladas, três cursos para a área de microinformática: "Introdução aos Microcomputadores", "Introdução ao BASIC" e "Aplicativos" com preços de 10, 15 e 20 ORTNs, respectivamente. Todos estes cursos têm duração de quatro dias, com aulas de segunda a quinta-feira, das 19 às 22hs. O endereço da MICROSHOP é Al. Lorena, 652, Jardim Paulista, São Paulo. Tel.: (011)282-2105.

• A ADVANCING — Produtos e Serviços em Informática dará início no dia 5 de maio aos cursos "Microprocessadores 8080/8085" e "Microprocessadores Z80A", ambos com 50 horas-aula. Maiores informações na própria ADVANCING, Rua dos Andradas, 1560, cj. 518 — Porto Alegre, Rio Grande do Sul. Tel.: (0512)26-8246.

• CDT Treinamento promoverá no mês de maio um curso de "Circuitos e Sistemas Digitais", num total de 25 horas e taxa de inscrição de Cr$ 80 mil. Maiores informações no próprio CDT, Av. Barão do Rio Branco, 882, Jardim Esplanada, São José dos Campos, SP. Tel.: (0123)21-9144, ramal 236.

• A SERVIMEC está formando nova turma para o seu curso de BASIC, que terá início no dia 25 de abril. As aulas serão ministradas às segundas, quartas e sextas-feiras, das 19: 30 às 22:30hs, num total de 52 horas. O preço de inscrição é Cr$ 40 mil. Maiores informações, na SERVIMEC, Rua Correa dos Santos, 26, Bom Retiro, São Paulo. Tel. (011) 222-1511.

• O Instituto Sullivan promove diversos cursos especializados na área de Informática: "Curso de linguagem BASIC para microcomputadores" (normal e avançado); "Linguagem Assembler (8080/86 e Z80)"; "Linguagens COBOL, FORTRAN e Pascal para microcomputadores". Com aulas práticas no CP-500, DGT-100, TRS-80, APPLE II e outros; turmas pela manhã, tarde e noite, especiais aos sábados e para empresas, o Sullivan também promove cursos para crianças de 8 a 13 anos. As vagas são limitadas e as reservas podem ser feitas pelo telefone (021) 295-0169 (plantão telefônico de 24h). O Instituto Sullivan fica na R. Siqueira Campos, 43, 7º andar, Copacabana, Rio de Janeiro, RJ.

• A BASE Tecnologia está oferecendo um curso de programação de micros com duração de 24 horas (8 sessões de 3 horas cada). O curso, que envolve a utilização prática de microcomputadores com discos, impressoras e com sistema operacional CP/M, custa Cr$ 35 mil (taxa de inscrição), havendo desconto para grupos de dois alunos que se inscrevam juntos. Informações pelo tel.: (021)227-4984 ou no local, Av. N. S. Copacabana, 1085, sl. 613, Copacabana, Rio de Janeiro, RJ.

• O Instituto Alcinda Fernandes promove regularmente o "Curso de Linguagem BASIC". Os alunos têm acesso direto ao DGT-100. As aulas serão práticas e o tempo de duração do curso é de 20 horas-aula, ministradas em 10 dias com turmas limitadas. Informações à R. Califórnia, 94, Bairro Sion, Belo Horizonte, MG.

• Para informar ao leitor sobre os cursos que estão sendo oferecidos, a revista recolhe informações em diversas instituições ou as recebe pelo correio. Portanto, não nos responsabilizamos por quaisquer alterações posteriormente efetuadas por estas instituições nos programas ou preços.

Veja como as características da gramática de linguagens de programação podem influenciar a estrutura de compiladores.

Compiladores e linguagens

Michael Stanton

Neste artigo apresentaremos algumas considerações sobre a definição de linguagens de programação de alto nível e as consequências que ela tem para a estrutura de compiladores feitos para estas linguagens. Veremos também como o projeto de um compilador poderá ser feito em função de características da definição da linguagem, de tal modo que a construção de um compilador se torne uma tarefa bastante automatizada e eficiente, reduzindo assim sua complexidade e seu custo.

A DEFINIÇÃO DE UMA LINGUAGEM

Para poder ser um veículo útil para a comunicação, é necessário que os usuários de uma linguagem concordem quanto à sua definição. No caso de uma linguagem de programação, o programador precisa saber quais programas ele poderá escrever e o que devem fazer estes programas. Sua visão da linguagem deverá coincidir com a visão do projetista da linguagem (que formula sua definição), bem como com a visão do implementador (que constrói o compilador, o qual traduz programas para uma forma executável num computador específico).

Uma dificuldade para as pessoas envolvidas com uma linguagem é a falta de uma notação geralmente aceita que descreva completamente todos os seus aspectos importantes. Porém, existe uma série de notações úteis que resolvem parcialmente o problema e que são de grande importância para a eficácia destas linguagens.

Assim, o que pode ser considerado um programa válido? Ao nível mais superficial, um programa é uma simples cadeia de caracteres escolhidos a partir de um alfabeto de caracteres. O alfabeto usado depende da linguagem, consistindo de letras (maiúsculas e/ou minúsculas), dígitos e um número variável de outros símbolos não alfanuméricos, como +, –, *, /, =, ' e $.

A SINTAXE

A sintaxe de uma linguagem é um conjunto de regras que determinam se uma dada cadeia é um programa válido. Porém, é bastante difícil especificar concisa e precisamente as regras de sintaxe de uma linguagem de programação, como também o é para o português. Felizmente existem notações que atendem parcialmente a esta questão, como gramáticas livres de contexto, que se mostram muito úteis para a construção de compiladores, como veremos a seguir.

No nível mais baixo, a cadeia que representa um programa poderá ser dividida em uma sequência de subcadeias chamadas de **tokens** (que vem do inglês e significa símbolos). Cada **token** é uma sequência de caracteres que têm significado coletivo. A maioria das linguagens trata como **tokens** os seguintes itens:

a. Constantes: 17, 3.14159
b. Identificadores: A, SORT, TOTAL
c. Operadores: +, –, =, **
d. Palavras reservadas: IF, DO, FUNCTION
e. Pontuação: parênteses, vírgula, ponto e vírgula

As estruturas sintáticas são agrupamentos de **tokens** e a maioria das linguagens contém expressões, que são sequências de **tokens** representando operandos e operadores. Estas expressões, junto com o operador de atribuição, símbolos de pontuação e palavras reservadas, são os elementos básicos a partir dos quais se formam as estruturas mais complexas da linguagem, tais como comandos, blocos e programas, em ordem crescente de complexidade (veja a figura 1).

Um programa sintaticamente correto deverá ser traduzido por um compilador. Porém, como sabemos o que faz este programa? Para traduzir qualquer linguagem, inclusive as línguas naturais, é imprescindível saber o significado daquilo que será traduzido.

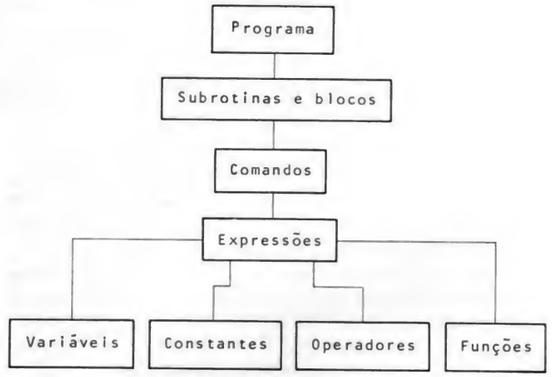


Figura 1 — Hierarquia de elementos de um programa

E temos ainda a semântica de uma linguagem, que são aquelas regras que dão significado às sentenças (ou programas) da linguagem. A semântica é muito mais difícil de definir que a sintaxe, e o método mais usado para defini-la é em termos do comportamento de uma máquina abstrata que executa os programas.

GRAMÁTICAS LIVRES DE CONTEXTO

Embora não exista uma notação conveniente para definir toda a sintaxe de uma linguagem de programação típica, existe uma classe de gramáticas, chamadas livres de contexto, que descrevem facilmente a maioria das características sintáticas de muitas linguagens de programação. Do ponto de vista do construtor de compiladores, estas gramáticas permitem a geração automática de analisadores sintáticos, reduzindo substancialmente a complexidade da tarefa de produzir um compilador.

Há diversas notações para escrever gramáticas livres de contexto, das quais as mais conhecidas em computação são BNF (do inglês Backus-Naur Form). Daremos um exemplo primeiro em BNF, e depois mostraremos o grafo de sintaxe correspondente. Consideremos a gramática de expressões envolvendo o identificador **a,** os operandos + e *, e parênteses. Uma gramática que mantém corretamente a prioridade de multiplicação sobre adição é:

1. E → E+T **3. T → T*F** **5. F → a**
2. E → T **4. T → F** **6. F → (E)**

Esta gramática consiste de um conjunto de ***regras de produção,*** ou ***produções,*** que exprimem os desdobramentos de ***símbolos não terminais*** (no caso, **E, T** e **F**) em termos de cadeias de símbolos não terminais e terminais (no caso, **a, (** e **)**). Os símbolos terminais pertencem ao alfabeto da linguagem descrita (ou gerada) pela gramática, que consiste neste caso de expressões tais como **a, a+a, a*(a+a)** e assim por diante.

Um dos símbolos não terminais (neste caso, **E**) é designado como símbolo inicial e uma cadeia na linguagem é gerada a partir dele por repetidas substituições de símbolos não terminais pelo lado direto de suas respectivas produções. Por exemplo, a aplicação sucessiva das regras 2, 3, 4, 5 e 6 gerará a seguinte sequência de **derivações:**

```
     2     3      4      5      5
E => T => T*F => F*F => a*F => a*a
```

A forma equivalente desta gramática usando grafos de sintaxe é ilustrada na figura 2. Neste caso, para cada símbolo não terminal corresponde um grafo dirigido, contendo nós terminais (círculos) e não terminais (quadrados). Uma cadeia de símbolos terminais é gerada caminhando nos grafos, começando por aquele que corresponde ao símbolo inicial. Ao encontrar um nó não terminal, suspendemos o grafo atual e começamos a caminhar no grafo que corresponde a este, voltando ao primeiro grafo quando sair do segundo.

Os grafos sintáticos para a definição de linguagens foram popularizados por Niklaus Wirth na sua definição de Pascal. Podemos notar que uma linguagem definida através desta notação é organizada hierarquicamente, onde as diferentes subestruturas da linguagem correspondem diretamente aos símbolos não terminais da gramática. Alguns dos não terminais usados na definição do Pascal são: programa, procedimento, bloco, parte de declarações de variáveis, parte de comandos, declarações de variável, comando, comando condicional, expressão e assim por diante. Portanto, estes não terminais estão intimamente ligados à estrutura de um programa e passam a fazer um papel importante na sua tradução.

Para descrever completamente a sintaxe de uma linguagem de programação são inadequadas as gramáticas livres de contexto, pelo fato das sentenças de um programa estarem sensíveis ao contexto em que aparecem. Para citar dois exemplos:

1 — A sintaxe (BNF) em Pascal do comando-condicional **IF expressão THEN comando ELSE comando,** onde **IF, THEN** e **ELSE** são símbolos terminais, e expressão e comando são não terminais. Porém, esta »

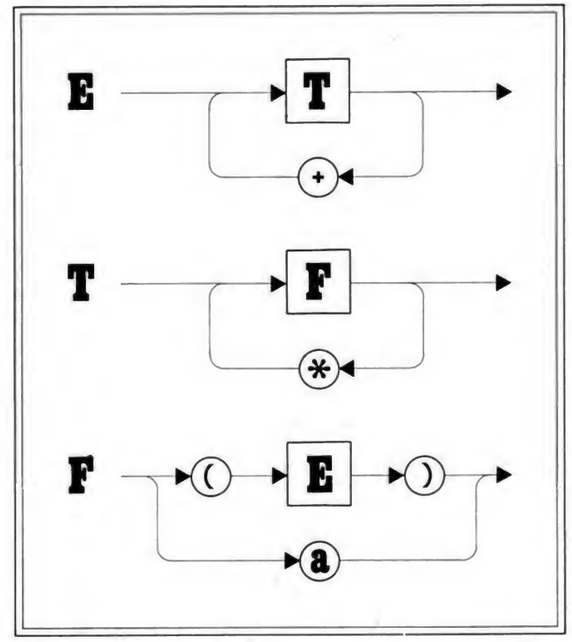


Figura 2 — Os grafos de sintaxe para expressões aritméticas simplificadas

forma somente será legítima se a expressão for do tipo **boolean** (lógico) e a determinação desta característica depende do contexto.

2 — Geralmente em Pascal todos os procedimentos (sub-rotinas) devem ser definidos antes do seu uso e numa chamada devem ser usados tantos argumentos quantos forem declarados na definição. Esta informação não pode ser especificada por uma gramática livre de contexto.

Nestes e em outros casos de sensibilidade ao contexto, a definição BNF da linguagem deve ser complementada por ***condições de contexto*** para definir completamente a sintaxe da linguagem.

Vejamos agora como é a estrutura geral de um compilador. Podemos reconhecer duas grandes fases do processo de compilação: ***análise*** e ***síntese.*** Na fase de ***análise*** é verificada a validade do programa, de acordo com a sintaxe da linguagem de programação. Na fase de ***síntese*** efetua-se a tradução para a linguagem objeto, implementando a semântica da linguagem de programação. Esta fase também é conhecida como geração de código objeto.

A ANÁLISE

Enquanto o usuário da linguagem de programação usa a definição da sintaxe da linguagem para guiar a criação do seu programa, o compilador deverá usar esta mesma definição para determinar se o programa está bem formado. O usuário ***sintetiza*** seu programa, enquanto o compilador o ***analisa.*** A conveniência das gramáticas livres de contexto deve-se à facilidade com a qual é possível escrever programas para fazer ***análise sintática*** (***parsing,*** em inglês) para elas.

Já foram descobertos uma série de métodos que permitem construir programas para o reconhecimento de linguagens livres de contexto. Os métodos mais recentes e mais convenientes são bastante simples na sua realização, consistindo de um programa iterativo com memória de pilha que caminha num grafo finito enquanto lê o programa sendo analisado.

O grafo finito é representado por tabelas de dados constantes dentro do programa de análise. Estes analisadores dirigidos por tabelas, que também são conhecidos como autômatos, não dependem da linguagem que está sendo analisada. Toda a informação específica à linguagem está codificada nas tabelas, que são geradas por um programa a partir da gramática (BNF ou grafos de sintaxe).

Os dois métodos mais conhecidos deste tipo são chamados LL(1) e LR(1) e existem grupos de pesquisa trabalhando com estes analisadores na USP (IME), na UFRJ (COPPE/Sistemas), na PUC/RJ (Informática) e na UFMG.

Como já observamos, a sintaxe de uma linguagem de programação é sensível ao contexto. Porém, podemos acrescentar aos analisadores de gramáticas livres de contexto a informação sobre o contexto da estrutura sintática que está sendo analisada. Num compilador, esta informação de contexto é incluída numa ***tabela de símbolos,*** que reúne a informação previamente declarada que determina o contexto corrente.

O analisador passa então a verificar que são satisfeitas as condições de contexto exigidas pela linguagem. Esta última análise é às vezes conhecida como ***análise semântica,*** embora estritamente faça parte da análise sintática do programa.

ANÁLISE LÉXICA

Normalmente o analisador sintático não trabalha diretamente com a forma original do programa e sim com uma forma semi-digerida desta.

Damos o nome de análise léxica àquela parte do compilador que lê o texto original do programa e o particiona em **tokens,** que são os símbolos terminais da gramática. Em particular, todos os identificadores são inseridos na ***tabela de símbolos*** e é passado no seu lugar, para o analisador sintático, um índice nesta tabela. Este pré-processamento da entrada simplifica o projeto do analisador sintático e é um excelente exemplo de boa modularização dentro do compilador.

SÍNTESE

Após a determinação da estrutura sintática de um programa, o compilador passa a traduzi-lo para o código objeto, que pode ser código de máquina, código de montagem (Assembler), código de uma máquina abstrata ou até uma outra linguagem de alto nível. Esta parte do compilador é aquela que menos se apóia em estudos teóricos, provavelmente devido à falta de uma notação conveniente para expressar as características da máquina objeto. Porém, podemos observar algumas características de compiladores em uso.

O mais importante é que a parte semântica do compilador depende fortemente da parte sintática e pode ser dita subordinada a esta no seguinte sentido: o compilador traduz diretamente para os conceitos da máquina objeto as estruturas sintáticas reconhecidas na fase de análise. Por exemplo, uma tradução de um comando condicional:

IF expressão THEN comando-1 ELSE comando-2

Seria equivalente ao seguinte trecho (em pseudo-linguagem):

IF expressão GOTO L1
comando-2
GOTO L2
onde **L1** é o **comando-1** e **L2** o **comando-2**

Em muitos compiladores, as rotinas responsáveis para a geração de código objeto estão embutidas nas rotinas de reconhecimento das diversas estruturas sintáticas. Às vezes são conhecidas como **rotinas semânticas.**

CONCLUSÃO

As propriedades muito convenientes de gramáticas livres de contexto podem ser facilmente aproveitadas para a construção de compiladores cuja estrutura central é erguida em cima do reconhecimento das estruturas sintáticas do programa. O analisador sintático é auxiliado pelos analisadores léxico e semântico e complementado pelas rotinas semânticas que geram o código objeto.

Falta apenas uma palavra sobre organização do programa do compilador. É possível e muito comum construir-se o compilador de um passo em que todas as fases são executadas concorrentemente, com os analisadores léxico e semântico e as rotinas de geração de código objeto sendo chamadas como sub-rotinas pela rotina principal de análise sintática.

Há, porém, muitos compiladores que adotam uma estrutura de passos múltiplos, onde cada fase processa o programa inteiro antes da fase subsequente. Neste caso, o programa é passado de uma fase à próxima em algum formato interno, como arquivos de código intermediário, ou de estruturas de dados na memória. Logicamente, para linguagens como Pascal, que são suscetíveis à compilação em um só passo, as duas alternativas são equivalentes. A primeira é mais rápida, porém usa mais memória.

Não devemos terminar esta breve excursão no campo de compiladores sem ao menos mencionar duas funções adicionais do compilador. O compilador só deve gerar código objeto para programas válidos. Para os programas inválidos, devem ser geradas mensagens de erro para todos os erros identificáveis na fonte.

Isto requer que o analisador sintático possa recuperar-se a partir de uma situação de erro, para poder continuar sua análise. O estudo de recuperação de erros de sintaxe é um campo fértil para a pesquisa.

Outra fase do compilador que possa existir é a de otimização do código gerado. Não consideramos isto aqui por falta de espaço, apesar de sua importância evidente.

Michael Stanton é coordenador de Pós-Graduação do Departamento de Informática da PUC/RJ, onde leciona e orienta pesquisas na área de software básico. Como coordenador da Comissão Especial para Linguagens e Sistemas de Programação da SBC — Sociedade Brasileira de Computação, participou da organização dos dois simpósios sobre o Desenvolvimento de Software Básico para Micros, realizados na USP e na PUC/RJ em 1982.

Mensagem de erro

No nº 16, matéria "Duas versões para o Jogo da Forca", página 13, segunda coluna, há um erro na linha nº 280 da listagem. A versão correta é a seguinte:

```
280 IF B$(F,I) = A$(D) THEN LET C$(D) = B$
```

No número 16, no **MICRO MERCADO**, algumas correções devem ser feitas nos seguintes equipamentos:

– **DGT-100** – É compatível com o TRS-80 Mod. I e não com o Mod. III, conforme saiu. Além disso, seu sistema operacional é o NEWDOS e não o TRSDOS.

– **HP-85** – Na versão brasileira, a capacidade máxima de memória RAM do aparelho é de 32 Kb e não de 64 Kb.

– **Fenix II** – Não está mais sendo fabricado.

– **QI 800** – Sua capacidade máxima de memória RAM é de 64 Kb e não de 56 Kb.

– **I 7010** – Após a edição estar nas bancas, a Itautec apresentou modificações no seu micro. Agora chamado apenas de I 7000, ele possui microprocessador NSC 800, da National, ao invés do 8085 da Intel.

De lá para cá novos micros foram lançados, como o JP-01 da Janper, o EGO da Softec, o Magic da Magnex, o JR da Sysdata, o Sistema 600 da Prológica, o TK85 da Microdigital, o PC 1211 da Sharp e o TRS-80 Mod. IV da Sayfi. Todos estarão presentes no próximo MICRO MERCADO, a sair em breve.

No número 17, na listagem do programa **Buck Rogers, (Guerra Espacial,** pág. 37) a linha 2040 saiu com um parêntese a menos. A linha correta é a seguinte:

```
2040 GOSUB1200:A=USR(2048+RND(101-5+RND(100)):RETURN
```

Procurando consolidar-se no mercado, a Computerland investe em três pontos estratégicos: hobby-lazer, assistência técnica e software.

Computerland, uma experiência que deu certo

Na matriz da Av. Angélica funciona o departamento de hobby e lazer e o clube de usuários de micro.

Inaugurada em fevereiro de 82, a Computerland, uma das pioneiras na comercialização de microcomputadores em São Paulo, já tem hoje claramente definidos seus objetivos e o melhor caminho a seguir. "Como uma das primeiras lojas do ramo, nos antecipamos à realidade nacional", comenta Arthur José Ribeiro Dias, sócio gerente da Computerland, "mas as dificuldades que enfrentamos nos serviram de base para encarar o verdadeiro mercado que agora se inicia. Em 83 vamos consolidar as experiências de 82".

Além da loja da Av. Angélica, 1996, a Computerland conta atualmente com uma filial em Campinas, onde funciona um show-room para desenvolvimento de software. Para breve já está prevista a inauguração de uma nova loja da rede, no bairro do Ibirapuera, em São Paulo.

Este ano, logo nos primeiros meses, a Computerland lançou uma série de novidades, entre elas um departamento de hobby e lazer e um clube onde o associado, mediante uma taxa, pode levar um microcomputador para casa juntamente com programas (hoje há cerca de 400 à disposição), e então ir se familiarizando com a máquina.

O departamento de hobby e lazer, segundo Arthur Ribeiro Dias, foi uma iniciativa para aproximar o grande público do microcomputador. "Este departamento é uma divisão dentro da loja na qual estão os equipamentos mais simples. Através dele visamos atrair, por exemplo, a dona de casa, que trará seu filho, seu marido, enfim, visamos desmistificar o micro perante todos".

A filosofia geral da Computerland é oferecer ao cliente as mesmas facilidades que um grande magazine oferece, tais como leasing, pagamento em até 24 meses etc. "Além disso", explica Arthur Ribeiro, "garantimos a assistência técnica especializada e grande variedade de software inédito para várias áreas".

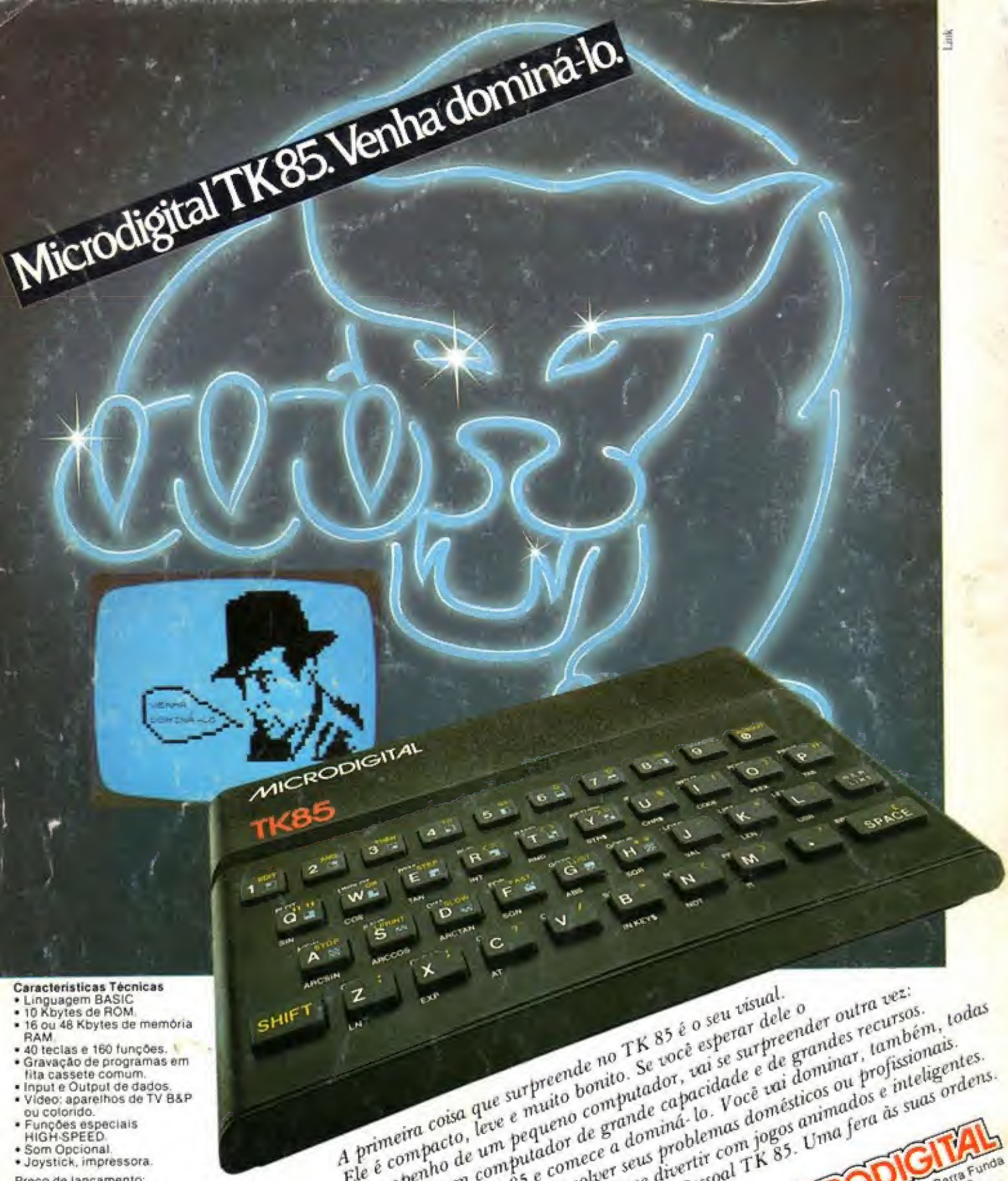
Quanto à assistência técnica, a loja lançou uma outra novidade: o Certificado de Garantia Computerland. Com ele, todos os equipamentos adquiridos têm seu prazo de garantia esticado para um ano, para casos de qualquer defeito e até mesmo reposição de peças. Com relação ao software, a filial de Campinas é voltada principalmente para o desenvolvimento de programas para equipamentos compatíveis com o Apple e o TRS-80. "Sentimos que o software é o grande negócio do futuro", afirma Arthur Ribeiro.

Como já estava previsto desde sua inauguração, a Computerland, ainda no primeiro semestre deste ano, passará a colocar em prática o esquema de franquia para abertura de lojas em outros estados. "Sabendo das dificuldades geradas pela falta de capital de giro na área de informática, eu como pessoa física pretendo entrar com 25% de capital em cada uma das lojas com nosso nome", esclarece Arthur Ribeiro. Além disso salientou que todos os novos sócios receberão também o pacote Computerland com toda a infra-estrutura da loja. Arthur Ribeiro Dias adiantou ainda que estão praticamente fechados contratos para abertura de lojas em Curitiba, Belo Horizonte, Rio de Janeiro, São José dos Campos, Porto Alegre e Ribeirão Preto.

Para o ano de 83, Arthur Ribeiro prevê que haverá uma tendência natural das pessoas se voltarem para o lazer doméstico, dado o momento de crise em que estamos vivendo. Na sua opinião, isto beneficiará em muito a área de informática.

Texto: Stela Lachtermacher
Foto : Nelson Jumo

Microdigital TK 85. Venha dominá-lo.



Características Técnicas
- Linguagem BASIC.
- 10 Kbytes de ROM.
- 16 ou 48 Kbytes de memória RAM.
- 40 teclas e 160 funções.
- Gravação de programas em fita cassete comum.
- Input e Output de dados.
- Vídeo: aparelhos de TV B&P ou colorido.
- Funções especiais HIGH-SPEED.
- Som Opcional.
- Joystick, impressora.

Preço de lançamento:
Cr$ 179.850,00 (16K)
Cr$ 249.850,00 (48K)
(Preço sujeito a alteração)

A primeira coisa que surpreende no TK 85 é o seu visual. Ele é compacto, leve e muito bonito. Se você esperar dele o desempenho de um pequeno computador, vai se surpreender outra vez: o TK 85 é um computador de grande capacidade e de grandes recursos. Acione o TK 85 e comece a dominá-lo. Você vai dominar, também, todas as situações. Resolver seus problemas domésticos ou profissionais. Vencer desafios e se divertir com jogos animados e inteligentes. Computador Pessoal TK 85. Uma fera às suas ordens.

MICRODIGITAL
Rua do Bosque, 1.234 - Barra Funda
CEP 01136 - Cx.P. 54.088 - São Paulo - SP
PABX 825-3355

REVENDEDORES: ARACAJU 224-1310 • **BELEM** 222-5122/226-0518 • **BELO HORIZONTE** 226-6336/225-3305/225-0644/201-7555 • **BLUMENAU** 22-1250 • **BRASÍLIA** 224-2777/225-4534/226-9201/226-4327/242-6344/242-5159 • **BRUSQUE** 55-0675 • **CAMPINAS** 32-3810/8-0822/32-4155/2-9930 • **CAMPO GRANDE** 383-6487/382-5332 • **CARUARU** 721-1273 • **CUIABA** 321-8119/321-7929 • **CURITIBA** 232-1750/224-6467/224-3422/243-1731/223-6944/233-8572/232-1196 • **DIVINÓPOLIS** 221-2942 • **FLORIANÓPOLIS** 23-1039 • **FORTALEZA** 226-4922/231-5249/231-0577/231-7013 • **FREDERICO WESTPHALEN** 344-1550 • **GOIÂNIA** 261-0333/224-0557 • **IJUI** 332-2740 • **ITAJUBA** 622-2088 • **LINS** 22-2428 • **LONDRINA** 22-4244/23-9674 • **MACEIÓ** 223-3979/221-6776 • **MANAUS** 237-1793 • **MOGI DAS CRUZES** 468-3779/208-6797 • **MURIAHÉ** 721-1593 • **NATAL** 222-3212/231-1055 • **NITERÓI** 722-6791 • **NOVO HAMBURGO** 93-1922/93-3800 • **PELOTAS** 24-5139 • **PORTO ALEGRE** 26-8246/21-4189/24-1411/22-3151/24-0311/21-6109/24-7746 • **PRESIDENTE PRUDENTE** 22-2788 • **RECIFE** 241-4310/224-8777/224-3436/224-4327 • **RESENDE** 54-1664 • **RIBEIRÃO PRETO** 636-0586/634-4715/635-1195 • **RIO DE JANEIRO** 267-1093/252-2050/253-3395/264-0143/259-1516/232-5948/591-3297/222-6088/267-1339/329-4869/228-2650/246-4824/239-5612/542-3849/62-8737 • **SALVADOR** 248-6666/235-4184/247-5717 • **SANTA MARIA** 221-7120 • **SANTO ANDRÉ** 455-4962/444-7375/454-9283 • **SANTOS** 4-1220/32-7045/35-1792/33-2230 • **SÃO CARLOS** 71-9424 • **SÃO JOÃO DA BOA VISTA** 22-3336 • **SÃO JOSÉ DOS CAMPOS** 22-3968/22-7311/22-8925/21-3135 • **SÃO PAULO** 853-0164/853-0448/239-4122/36-6961/61-4049/881-1149/258-3954/212-9004/282-2105/212-3888/545-4769/227-3022/864-8200/222-1511/259-2600/282-6609/813-4555/814-3663/826-1499/521-3779/270-7442/210-7681/813-4031 • **SOROCABA** 32-9988 • **TAUBATÉ** 31-4137 • **UBERABA** 333-1091 • **UBERLÂNDIA** 234-8796 • **VIÇOSA** 891-1790/891-2258 • **MARILIA** 33-4109